ML Pipeline Preparation

July 21, 2020

1 ML Pipeline Preparation

Follow the instructions below to help you create your ML pipeline. ### 1. Import libraries and load data from database. - Import Python libraries - Load dataset from database with `read_sql_table` - Define feature and target variables X and Y

```
In [1]: # import libraries
        import sys

        import nltk
        nltk.download(['punkt', 'wordnet', 'averaged_perceptron_tagger'])
        nltk.download('stopwords')

        import re
        import numpy as np
        import pandas as pd
        import pickle
        from sqlalchemy import create_engine

        from nltk.tokenize import word_tokenize
        from nltk.stem import WordNetLemmatizer
        from nltk.corpus import stopwords

        from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
        from sklearn.model_selection import train_test_split, GridSearchCV
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
        from sklearn.multioutput import MultiOutputClassifier
        from sklearn.pipeline import Pipeline, FeatureUnion
        from sklearn.base import BaseEstimator, TransformerMixin
        from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
```

```
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
In [2]: # load data from database
        engine = create_engine('sqlite:///{}'.format('DisasterResponse.db'))
        df = pd.read_sql_table('DisasterResponse', con=engine)

        columns = df.columns[4:]
        X = df['message'].values
        Y = df[df.columns[4:]].values
```

```
In [3]: X[:5]
```

```
Out[3]: array(['Weather update - a cold front from Cuba that could pass over Haiti',
               'Is the Hurricane over or is it not over',
               'Looking for someone but no name',
               'UN reports Leogane 80-90 destroyed. Only Hospital St. Croix functioning. Needs s
               'says: west side of Haiti, rest of the country today and tonight'], dtype=object)
```

```
In [4]: Y[:5]
```

```
Out[4]: array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
               [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0],
               [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
               [1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
                0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
               [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

1.0.1 2. Write a tokenization function to process your text data

```
In [5]: def tokenize(text):
            # Normalize text
            text = re.sub(r"[^a-zA-Z0-9]", " ", text.lower())

            # Tokenize text (split text into words)
            tokens = word_tokenize(text)

            # Remove stop words
            words = [word for word in tokens if word not in stopwords.words('english')]

            # Lemmatization (reduce words to their root form)
            lemmed = [WordNetLemmatizer().lemmatize(w) for w in words]

            return lemmed
```

1.0.2 3. Build a machine learning pipeline

This machine pipeline should take in the `message` column as input and output classification results on the other 36 categories in the dataset. You may find the MultiOutputClassifier helpful for predicting multiple target variables.

```
In [6]: pipeline = Pipeline([
                ('vect', CountVectorizer(tokenizer=tokenize)),
                ('tfidf', TfidfTransformer()),
                ('clf', RandomForestClassifier(class_weight='balanced'))
            ])
```

1.0.3 4. Train pipeline

- Split data into train and test sets
- Train pipeline

```
In [7]: # Split data into train and test sets
        X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state=0)

In [8]: # Train pipeline
        pipeline.fit(X_train, y_train)

Out[8]: Pipeline(memory=None,
             steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='strict
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=None, min_df=1,
                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                strip...imators=10, n_jobs=1, oob_score=False, random_state=None,
                    verbose=0, warm_start=False))])
```

1.0.4 5. Test your model

Report the f1 score, precision and recall for each output category of the dataset. You can do this by iterating through the columns and calling sklearn's `classification_report` on each.

```
In [9]: pred = pipeline.predict(X_test)

        precisions, recalls, f1s = [], [], []
        for idx,column in enumerate(columns):
            precisions.append(precision_score(y_test[:,idx], pred[:,idx]))
            recalls.append(recall_score(y_test[:,idx], pred[:,idx]))
            f1s.append(f1_score(y_test[:,idx], pred[:,idx]))

        report = pd.DataFrame({'Categories': columns, 'Precision': precisions, 'Recall': recalls
        report

/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
```

```
  'precision', 'predicted', average, warn_for)
```

Out[9]:

	Categories	Precision	Recall	F1
0	related	0.766331	1.000000	0.867709
1	request	0.296566	0.258856	0.276431
2	offer	0.000000	0.000000	0.000000
3	aid_related	0.420394	1.000000	0.591940
4	medical_help	0.080073	1.000000	0.148274
5	medical_products	0.052607	1.000000	0.099956
6	search_and_rescue	0.027698	0.159341	0.047193
7	security	0.015038	0.017391	0.016129
8	military	0.000000	0.000000	0.000000
9	child_alone	0.000000	0.000000	0.000000
10	water	0.065473	1.000000	0.122900
11	food	0.113744	0.998658	0.204227
12	shelter	0.090576	1.000000	0.166106
13	clothing	0.010606	0.069307	0.018397
14	money	0.018868	0.007246	0.010471
15	missing_people	0.010584	0.985507	0.020942
16	refugees	0.052288	0.110092	0.070901
17	death	0.044906	1.000000	0.085952
18	other_aid	0.150000	0.026966	0.045714
19	infrastructure_related	0.062433	1.000000	0.117529
20	transport	0.044628	0.996587	0.085430
21	buildings	0.050985	1.000000	0.097023
22	electricity	0.019206	0.967742	0.037665
23	tools	0.005105	0.968750	0.010157
24	hospitals	0.011664	0.946667	0.023044
25	shops	0.000000	0.000000	0.000000
26	aid_centers	0.011452	0.986667	0.022640
27	other_infrastructure	0.021739	0.003623	0.006211
28	weather_related	0.286935	1.000000	0.445920
29	floods	0.121495	0.217472	0.155896
30	storm	0.171271	0.049919	0.077307
31	fire	0.000000	0.000000	0.000000
32	earthquake	0.098275	0.976415	0.178577
33	cold	0.000000	0.000000	0.000000
34	other_weather	0.102564	0.010667	0.019324
35	direct_report	0.000000	0.000000	0.000000

1.0.5 6. Improve your model

Use grid search to find better parameters.

In [12]:
```
parameters = {
                'tfidf__use_idf': (True, False),
                'clf__n_estimators': [10, 20]
```

```
    }

cv = GridSearchCV(pipeline, param_grid=parameters)
```

1.0.6 7. Test your model

Show the accuracy, precision, and recall of the tuned model.

Since this project focuses on code quality, process, and pipelines, there is no minimum performance metric needed to pass. However, make sure to fine tune your models for accuracy, precision and recall to make your project stand out - especially for your portfolio!

```
In [13]: cv.fit(X_train, y_train)

Out[13]: GridSearchCV(cv=None, error_score='raise',
                estimator=Pipeline(memory=None,
              steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='stric
                 dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                 lowercase=True, max_df=1.0, max_features=None, min_df=1,
                 ngram_range=(1, 1), preprocessor=None, stop_words=None,
                 strip...imators=10, n_jobs=1, oob_score=False, random_state=None,
                     verbose=0, warm_start=False))]),
                fit_params=None, iid=True, n_jobs=1,
                param_grid={'tfidf__use_idf': (True, False), 'clf__n_estimators': [10, 20]},
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring=None, verbose=0)

In [14]: pred = cv.predict(X_test)

         precisions, recalls, f1s = [], [], []
         for idx,column in enumerate(columns):
             precisions.append(precision_score(y_test[:,idx], pred[:,idx]))
             recalls.append(recall_score(y_test[:,idx], pred[:,idx]))
             f1s.append(f1_score(y_test[:,idx], pred[:,idx]))

         report = pd.DataFrame({'Categories': columns, 'Precision': precisions, 'Recall': recall
         report

/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
```

Out[14]:

	Categories	Precision	Recall	F1
0	related	0.766331	1.000000	0.867709
1	request	0.216894	0.636694	0.323563
2	offer	0.000000	0.000000	0.000000
3	aid_related	0.420330	1.000000	0.591876
4	medical_help	0.080012	1.000000	0.148169

5	medical_products	0.052640	1.000000	0.100015
6	search_and_rescue	0.018519	0.005495	0.008475
7	security	0.031930	0.191304	0.054726
8	military	0.000000	0.000000	0.000000
9	child_alone	0.000000	0.000000	0.000000
10	water	0.064232	0.936916	0.120222
11	food	0.113775	1.000000	0.204305
12	shelter	0.090368	0.998314	0.165733
13	clothing	0.142857	0.009901	0.018519
14	money	0.028470	0.173913	0.048930
15	missing_people	0.010519	0.913043	0.020799
16	refugees	0.042373	0.389908	0.076439
17	death	0.044892	1.000000	0.085927
18	other_aid	0.130094	0.484270	0.205092
19	infrastructure_related	0.062424	1.000000	0.117512
20	transport	0.045164	0.941980	0.086196
21	buildings	0.051000	1.000000	0.097051
22	electricity	0.019021	1.000000	0.037333
23	tools	0.004567	0.843750	0.009085
24	hospitals	0.011759	0.933333	0.023225
25	shops	0.000000	0.000000	0.000000
26	aid_centers	0.011611	0.933333	0.022936
27	other_infrastructure	0.040129	0.315217	0.071195
28	weather_related	0.286935	1.000000	0.445920
29	floods	0.100788	0.570632	0.171317
30	storm	0.128092	0.558776	0.208408
31	fire	0.142857	0.025974	0.043956
32	earthquake	0.097173	1.000000	0.177134
33	cold	0.000000	0.000000	0.000000
34	other_weather	0.073873	0.362667	0.122744
35	direct_report	0.000000	0.000000	0.000000

1.0.7 8. Try improving your model further. Here are a few ideas:

- try other machine learning algorithms
- add other features besides the TF-IDF

In [15]:
```
# Build a machine learning pipeline

pipeline_abc = Pipeline([
    ('vect', CountVectorizer(tokenizer=tokenize)),
    ('tfidf', TfidfTransformer()),
    ('clf', MultiOutputClassifier(AdaBoostClassifier(base_estimator=DecisionTreeClassif
    ))
])

parameters_abc = {
    'tfidf__use_idf': (True, False),
```

```
        'vect__ngram_range': ((1, 1), (1, 2))
    }

    cv_abc = GridSearchCV(estimator=pipeline_abc, param_grid=parameters_abc, cv=3, scoring=
```

```
In [16]: # Fit the model
         cv_abc.fit(X_train, y_train)

         # Best parameters set
         cv_abc.best_params_
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1137: UndefinedMetricWa
  'recall', 'true', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1137: UndefinedMetricWa
  'recall', 'true', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1137: UndefinedMetricWa
  'recall', 'true', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1137: UndefinedMetricWa
  'recall', 'true', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1137: UndefinedMetricWa
  'recall', 'true', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1137: UndefinedMetricWa
  'recall', 'true', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1137: UndefinedMetricWa
  'recall', 'true', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1137: UndefinedMetricWa
  'recall', 'true', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1137: UndefinedMetricWa
  'recall', 'true', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
```

```
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1137: UndefinedMetricWa
  'recall', 'true', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1137: UndefinedMetricWa
  'recall', 'true', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1137: UndefinedMetricWa
  'recall', 'true', average, warn_for)
```

Out[16]: {'tfidf__use_idf': False, 'vect__ngram_range': (1, 2)}

In [17]:
```
# Make predictions
pred = cv_abc.predict(X_test)

precisions, recalls, f1s = [], [], []
for idx,column in enumerate(columns):
    precisions.append(precision_score(y_test[:,idx], pred[:,idx]))
    recalls.append(recall_score(y_test[:,idx], pred[:,idx]))
    f1s.append(f1_score(y_test[:,idx], pred[:,idx]))

report = pd.DataFrame({'Categories': columns, 'Precision': precisions, 'Recall': recall
report
```

Out[17]:

	Categories	Precision	Recall	F1
0	related	0.770346	0.984067	0.864189
1	request	0.449664	0.121708	0.191565
2	offer	0.000000	0.000000	0.000000
3	aid_related	0.484340	0.157226	0.237390
4	medical_help	0.062500	0.001908	0.003704
5	medical_products	0.000000	0.000000	0.000000
6	search_and_rescue	0.000000	0.000000	0.000000
7	security	0.000000	0.000000	0.000000
8	military	0.000000	0.000000	0.000000
9	child_alone	0.000000	0.000000	0.000000
10	water	0.000000	0.000000	0.000000
11	food	0.297872	0.018792	0.035354
12	shelter	0.125000	0.003373	0.006568
13	clothing	0.142857	0.009901	0.018519
14	money	0.000000	0.000000	0.000000
15	missing_people	0.000000	0.000000	0.000000
16	refugees	0.000000	0.000000	0.000000
17	death	0.000000	0.000000	0.000000
18	other_aid	0.086957	0.004494	0.008547
19	infrastructure_related	0.166667	0.004890	0.009501

```
20                transport   0.166667  0.003413  0.006689
21                buildings   0.307692  0.011976  0.023055
22              electricity   0.000000  0.000000  0.000000
23                    tools   0.000000  0.000000  0.000000
24                hospitals   0.000000  0.000000  0.000000
25                    shops   0.000000  0.000000  0.000000
26              aid_centers   0.000000  0.000000  0.000000
27     other_infrastructure   0.000000  0.000000  0.000000
28          weather_related   0.565147  0.184574  0.278268
29                   floods   0.066667  0.001859  0.003617
30                    storm   0.394231  0.066023  0.113103
31                     fire   0.000000  0.000000  0.000000
32               earthquake   0.563291  0.139937  0.224181
33                     cold   0.000000  0.000000  0.000000
34            other_weather   0.000000  0.000000  0.000000
35            direct_report   0.424028  0.095314  0.155642
```

1.0.8 9. Export your model as a pickle file

```
In [18]: with open ('classifier', 'wb') as f:
             pickle.dump(cv, f)
```

1.0.9 10. Use this notebook to complete `train.py`

Use the template file attached in the Resources folder to write a script that runs the steps above to create a database and export a model based on a new dataset specified by the user.

```
In [ ]:
```