ML Pipeline Preparation

April 16, 2019

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 pandas as pd
import numpy as np
from sqlalchemy import create_engine
import sys
import re
import pickle
import nltk
from nltk.tokenize import word_tokenize,sent_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer, TfidfVect
from sklearn.multioutput import MultiOutputClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection  import GridSearchCV

%matplotlib inline
```

In [2]:
```
nltk.download(['punkt', 'wordnet', 'stopwords'])
```

```
[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 stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
Out[2]: True
```

```
In [3]: # load data from database
        engine = create_engine('sqlite:///DisasterResponse.db')
        df = pd.read_sql_table('labeled_data_messages', engine)
```

```
In [4]: # drop nan values
        df.dropna(axis=0, how = 'any', inplace = True)

        X = df['message']
        y = df.iloc[:,4:].astype(int)
```

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

```
In [5]: def tokenize(text):

            text = re.sub(r"[^a-zA-Z0-9]", " ", text.lower())

            tokens = word_tokenize(text)
            lemmatizer = WordNetLemmatizer()

            clean_tokens = []
            for tok in tokens:
                clean_tok = lemmatizer.lemmatize(tok).lower().strip()
                clean_tokens.append(clean_tok)

            return clean_tokens
```

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', MultiOutputClassifier(RandomForestClassifier()))
        ])
```

1.0.3 4. Train pipeline

- Split data into train and test sets
- Train pipeline

In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)

pipeline.fit(X_train, y_train)
```

Out[7]:
```
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...oob_score=False, random_state=None, verbose=0,
            warm_start=False),
           n_jobs=1))])
```

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 [8]:
```
def generate_report(y_test, y_pred):

    metrics = []
    for i, column in enumerate(y.columns.values):
        accuracy = accuracy_score(y_test[:,i], y_pred[:,i])
        precision = precision_score(y_test[:,i], y_pred[:,i], average='micro')
        recall = recall_score(y_test[:,i], y_pred[:,i], average='micro')
        f1 = f1_score(y_test[:,i], y_pred[:,i], average='micro')

        metrics.append([accuracy, precision, recall, f1])

    df = pd.DataFrame(data = np.array(metrics), index=y.columns.values, columns=['Accura
    return df
```

In [9]:
```
# Evaluate training set
y_train_pred = pipeline.predict(X_train)
```

In [10]:
```
generate_report(np.array(y_train), y_train_pred)
```

Out[10]:

	Accuracy	Precision	Recall	F1 score
related	0.984896	0.984896	0.984896	0.984896
request	0.967560	0.967560	0.967560	0.967560
offer	0.999737	0.999737	0.999737	0.999737
aid_related	0.972813	0.972813	0.972813	0.972813
medical_help	0.984502	0.984502	0.984502	0.984502
medical_products	0.991988	0.991988	0.991988	0.991988
search_and_rescue	0.993827	0.993827	0.993827	0.993827
security	0.994484	0.994484	0.994484	0.994484
military	0.998687	0.998687	0.998687	0.998687
child_alone	1.000000	1.000000	1.000000	1.000000
water	0.980299	0.980299	0.980299	0.980299

food	0.968348	0.968348	0.968348	0.968348
shelter	0.976885	0.976885	0.976885	0.976885
clothing	0.998030	0.998030	0.998030	0.998030
money	0.996060	0.996060	0.996060	0.996060
missing_people	0.997111	0.997111	0.997111	0.997111
refugees	0.995797	0.995797	0.995797	0.995797
death	0.993958	0.993958	0.993958	0.993958
other_aid	0.970186	0.970186	0.970186	0.970186
infrastructure_related	0.991988	0.991988	0.991988	0.991988
transport	0.995009	0.995009	0.995009	0.995009
buildings	0.987786	0.987786	0.987786	0.987786
electricity	0.998030	0.998030	0.998030	0.998030
tools	0.999081	0.999081	0.999081	0.999081
hospitals	0.998818	0.998818	0.998818	0.998818
shops	0.999475	0.999475	0.999475	0.999475
aid_centers	0.997636	0.997636	0.997636	0.997636
other_infrastructure	0.994221	0.994221	0.994221	0.994221
weather_related	0.970581	0.970581	0.970581	0.970581
floods	0.994878	0.994878	0.994878	0.994878
storm	0.990806	0.990806	0.990806	0.990806
fire	0.999343	0.999343	0.999343	0.999343
earthquake	0.980299	0.980299	0.980299	0.980299
cold	0.998949	0.998949	0.998949	0.998949
other_weather	0.995535	0.995535	0.995535	0.995535
direct_report	0.971763	0.971763	0.971763	0.971763

In [11]:
```
y_test_pred = pipeline.predict(X_test)
```

In [12]:
```
generate_report(np.array(y_test), y_test_pred)
```

Out[12]:

	Accuracy	Precision	Recall	F1 score
related	0.616384	0.616384	0.616384	0.616384
request	0.630957	0.630957	0.630957	0.630957
offer	0.998818	0.998818	0.998818	0.998818
aid_related	0.595510	0.595510	0.595510	0.595510
medical_help	0.939740	0.939740	0.939740	0.939740
medical_products	0.962190	0.962190	0.962190	0.962190
search_and_rescue	0.979913	0.979913	0.979913	0.979913
security	0.985821	0.985821	0.985821	0.985821
military	0.996849	0.996849	0.996849	0.996849
child_alone	1.000000	1.000000	1.000000	1.000000
water	0.922410	0.922410	0.922410	0.922410
food	0.845609	0.845609	0.845609	0.845609
shelter	0.886963	0.886963	0.886963	0.886963
clothing	0.992123	0.992123	0.992123	0.992123
money	0.986609	0.986609	0.986609	0.986609
missing_people	0.994092	0.994092	0.994092	0.994092
refugees	0.984246	0.984246	0.984246	0.984246

```
death                   0.971642  0.971642  0.971642  0.971642
other_aid               0.839307  0.839307  0.839307  0.839307
infrastructure_related  0.971642  0.971642  0.971642  0.971642
transport               0.982670  0.982670  0.982670  0.982670
buildings               0.961008  0.961008  0.961008  0.961008
electricity             0.993698  0.993698  0.993698  0.993698
tools                   0.997243  0.997243  0.997243  0.997243
hospitals               0.995274  0.995274  0.995274  0.995274
shops                   0.996455  0.996455  0.996455  0.996455
aid_centers             0.992911  0.992911  0.992911  0.992911
other_infrastructure    0.985033  0.985033  0.985033  0.985033
weather_related         0.827885  0.827885  0.827885  0.827885
floods                  0.971642  0.971642  0.971642  0.971642
storm                   0.963765  0.963765  0.963765  0.963765
fire                    0.995274  0.995274  0.995274  0.995274
earthquake              0.907050  0.907050  0.907050  0.907050
cold                    0.993698  0.993698  0.993698  0.993698
other_weather           0.979519  0.979519  0.979519  0.979519
direct_report           0.619141  0.619141  0.619141  0.619141
```

1.0.5 6. Improve your model

Use grid search to find better parameters.

In [13]:
```
RandomForestClassifier().get_params()
```

Out[13]:
```
{'bootstrap': True,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 10,
 'n_jobs': 1,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [14]:
```
parameters = {
    'vect__min_df': [1, 5],
    'tfidf__use_idf':[True, False],
    'clf__estimator__n_estimators':[10, 25],
    'clf__estimator__min_samples_split':[2, 5, 10]
```

```
}

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 [15]: cv_model = cv.fit(X_train, y_train)
```

```
In [16]: cv.best_params_
```

```
Out[16]: {'clf__estimator__min_samples_split': 2,
          'clf__estimator__n_estimators': 10,
          'tfidf__use_idf': False,
          'vect__min_df': 1}
```

```
In [17]: y_test_pred_cv = cv.predict(X_test)
```

```
In [18]: generate_report(np.array(y_test), y_test_pred_cv)
```

Out[18]:

	Accuracy	Precision	Recall	F1 score
related	0.609295	0.609295	0.609295	0.609295
request	0.636077	0.636077	0.636077	0.636077
offer	0.998818	0.998818	0.998818	0.998818
aid_related	0.596692	0.596692	0.596692	0.596692
medical_help	0.942497	0.942497	0.942497	0.942497
medical_products	0.965735	0.965735	0.965735	0.965735
search_and_rescue	0.979519	0.979519	0.979519	0.979519
security	0.986609	0.986609	0.986609	0.986609
military	0.996849	0.996849	0.996849	0.996849
child_alone	1.000000	1.000000	1.000000	1.000000
water	0.924774	0.924774	0.924774	0.924774
food	0.848365	0.848365	0.848365	0.848365
shelter	0.886570	0.886570	0.886570	0.886570
clothing	0.992123	0.992123	0.992123	0.992123
money	0.986609	0.986609	0.986609	0.986609
missing_people	0.994092	0.994092	0.994092	0.994092
refugees	0.984640	0.984640	0.984640	0.984640
death	0.971249	0.971249	0.971249	0.971249
other_aid	0.851122	0.851122	0.851122	0.851122
infrastructure_related	0.971249	0.971249	0.971249	0.971249
transport	0.981883	0.981883	0.981883	0.981883
buildings	0.961008	0.961008	0.961008	0.961008
electricity	0.993698	0.993698	0.993698	0.993698
tools	0.997243	0.997243	0.997243	0.997243

```
hospitals               0.995274   0.995274  0.995274  0.995274
shops                   0.996455   0.996455  0.996455  0.996455
aid_centers             0.992911   0.992911  0.992911  0.992911
other_infrastructure    0.985033   0.985033  0.985033  0.985033
weather_related         0.827491   0.827491  0.827491  0.827491
floods                  0.972824   0.972824  0.972824  0.972824
storm                   0.964553   0.964553  0.964553  0.964553
fire                    0.995274   0.995274  0.995274  0.995274
earthquake              0.907050   0.907050  0.907050  0.907050
cold                    0.993698   0.993698  0.993698  0.993698
other_weather           0.979913   0.979913  0.979913  0.979913
direct_report           0.625049   0.625049  0.625049  0.625049
```

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 [19]: pipeline2 = Pipeline([
                 ('vect', CountVectorizer(tokenizer=tokenize)),
                 ('tfidf', TfidfTransformer()),
                 ('clf', MultiOutputClassifier(AdaBoostClassifier()))
         ])

         parameters2 = {'vect__min_df': [5],
                       'tfidf__use_idf':[True],
                       'clf__estimator__learning_rate': [0.5, 1],
                       'clf__estimator__n_estimators':[10, 25]}

         cv2 = GridSearchCV(pipeline2, param_grid=parameters2)

In [20]: AdaBoostClassifier().get_params()

Out[20]: {'algorithm': 'SAMME.R',
          'base_estimator': None,
          'learning_rate': 1.0,
          'n_estimators': 50,
          'random_state': None}

In [21]: cv2_model = cv2.fit(X_train, y_train)

In [22]: y_test_pred_cv2 = cv2.predict(X_test)

In [23]: generate_report(np.array(y_test), y_test_pred_cv2)

Out[23]:                         Accuracy  Precision    Recall  F1 score
         related                0.658133   0.658133  0.658133  0.658133
         request                0.643954   0.643954  0.643954  0.643954
         offer                  0.998425   0.998425  0.998425  0.998425
```

```
aid_related             0.612052   0.612052  0.612052  0.612052
medical_help            0.944072   0.944072  0.944072  0.944072
medical_products        0.966916   0.966916  0.966916  0.966916
search_and_rescue       0.979913   0.979913  0.979913  0.979913
security                0.987003   0.987003  0.987003  0.987003
military                0.996455   0.996455  0.996455  0.996455
child_alone             1.000000   1.000000  1.000000  1.000000
water                   0.927137   0.927137  0.927137  0.927137
food                    0.852304   0.852304  0.852304  0.852304
shelter                 0.894840   0.894840  0.894840  0.894840
clothing                0.992123   0.992123  0.992123  0.992123
money                   0.986215   0.986215  0.986215  0.986215
missing_people          0.994092   0.994092  0.994092  0.994092
refugees                0.984640   0.984640  0.984640  0.984640
death                   0.972036   0.972036  0.972036  0.972036
other_aid               0.859000   0.859000  0.859000  0.859000
infrastructure_related  0.971249   0.971249  0.971249  0.971249
transport               0.982670   0.982670  0.982670  0.982670
buildings               0.963765   0.963765  0.963765  0.963765
electricity             0.993698   0.993698  0.993698  0.993698
tools                   0.997243   0.997243  0.997243  0.997243
hospitals               0.994880   0.994880  0.994880  0.994880
shops                   0.995668   0.995668  0.995668  0.995668
aid_centers             0.992517   0.992517  0.992517  0.992517
other_infrastructure    0.985033   0.985033  0.985033  0.985033
weather_related         0.840488   0.840488  0.840488  0.840488
floods                  0.976763   0.976763  0.976763  0.976763
storm                   0.965735   0.965735  0.965735  0.965735
fire                    0.995274   0.995274  0.995274  0.995274
earthquake              0.912564   0.912564  0.912564  0.912564
cold                    0.993698   0.993698  0.993698  0.993698
other_weather           0.980307   0.980307  0.980307  0.980307
direct_report           0.641197   0.641197  0.641197  0.641197
```

1.0.8 9. Export your model as a pickle file

```
In [24]: with open('classifer.pkl', 'wb') as f:
             pickle.dump(cv2, 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 [ ]:
```