Bienvenidos a la Wiki de PLNLangDetection!

Este proyecto ha sido desarrollado por Sergio Rodríguez Calvo para ser evaluado en la asignatura de Procesamiento del Lenguaje Natural del Máster Universitario en Lógica, Computación e Inteligencia Artificial de la Universidad de Sevilla.

Los requisitos quedan explicados (en inglés) en el siguiente link. El objetivo aquí es explicar la solución al problema.

Para la implementación de la solución se ha empleado Python3.6.

Las librerías utilizadas se encuentran en requirements.txt. Su instalación se realiza mediante pip3, de la siguiente manera:

```
pip3 install -r requirements.txt
```

Previamente, se necesitan datos con los que entrenar al sistema y con los que realizar su evaluación.

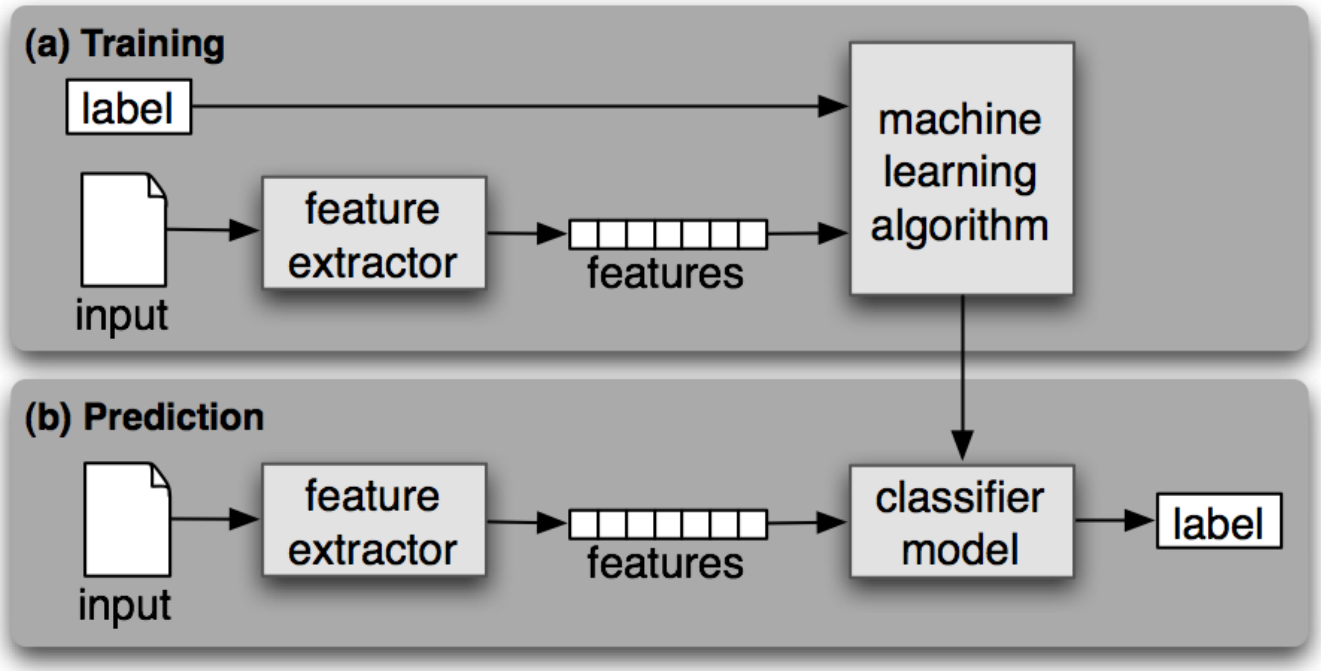
Para los datos de entrenamiento, se ha utilizado Europarl, concretamente los datos provenientes del siguiente link.

Para los datos de evaluación, se ha realizado un desarrollo propio, el cual realiza una descarga de ficheros de texto procesados por idioma y tema. Más información en el siguiente link. En el propio repositorio, se encuentra un ejemplo de fichero topics.json, mediante el cual se indica al script los idiomas que se desean contemplar, junto al nombre del directorio objetivo y los temas a descargar.

Para desarrollar el sistema, y cumplir con los requisitos, se ha optado por la creación de dos scripts: * Script para entrenamiento, train.py. * Script para evaluación, classify.py.

Ambos scripts se explican a continuación:

Un esquema general se puede observar a continuación, donde (a) es una aproximación a train.py, y (b) una aproximación a classify.py.



train.py

Este script es el responsable de realizar el entrenamiento y generar un modelo que se podrá utilizar posteriormente en classify.py para realizar evaluaciones, evitando la necesidad de entrenar el sistema una y otra vez.

Este script se ejecuta de la siguiente forma:

```
~$ python3 train.py languages.txt europarl_raw/
```

En este caso, se utiliza el directorio que se obtiene desde europarl, *europarl_raw/*, como se indica anteriormente, el cual contiene los mismos textos obtenidos a partir del diario de sesiones del parlamento Europeo y traducidos a los idiomas oficiales de la Unión Europea, separados por en directorios por idiomas. También es necesario un fichero *languages.txt*, donde se indica que lenguajes se quiere tener en cuenta a la hora de realizar el entrenamiento.

Es importante que para cada lenguaje indicado en *languages.txt* exista un directorio fuente dentro de *europarl_raw/* con el mismo nombre.

Una vez ejecutado el sistema produce un fichero llamado *modle.pickle*, que contiene el modelo necesario para realizar la evaluación en el script que se comentará posteriormente.

Antes de pasar a comentar el código, hay que tener en cuenta que el número de ficheros de europarl es elevado, como se indica en el siguiente link, por lo que la ejecución del sistema puede tomar varios minutos, y que puede variar dependiendo del rendimiento de la máquina donde se realiza la ejecución.

En cuanto a la implementación, en primer lugar se pretende obtener los parámetros introducidos en la llamada por consola:

```
parser = argparse.ArgumentParser(description='Train model for document
classification')
parser.add_argument('languages', metavar='M', type=str, help='Languages for
training')
parser.add_argument('dataset', metavar='D', type=str, help='Folder containing the
documents within the subfolders')
args = parser.parse_args()
```

Para ello se hace uso de la librería argparse, la cual simplifica esta tarea y ayuda controlando errores en la llamada.

El siguiente paso consiste en obtener la lista de lenguajes que se desea utilizar:

```
languages_for_training = list(open(args.languages,'r'))[0].rstrip().split(',')
#Remove \n
```

Y a coninuación, se realiza la busqueda de los ficheros de *europarl_raw/*, así como, un filtrado de los idiomas que se desean emplear (*get_directories*), como se puede observar a continuación:

```
files = find_files(args.dataset, languages_for_training)

def get_directories(data_path, languages):
    directories = [d for d in listdir(data_path) if not d.startswith('.')]
    for language in languages:
        if language not in directories:
            print("Cannot find '" + language + "'' directory into filesystem.")
            print("Please, check if exists '" + language + "'' into europarl_raw
directory or remove from languages.txt.")
            languages = list()
            break
    return languages

def find_files(data_path, languages_for_training):
    files = dict()
    languages = get_directories(data_path, languages_for_training)
    for lang in languages:
        files[lang] = [f for f in listdir('%s/%s/'%(data_path,lang)) if not
f.startswith('.')]
        print("%s : %i ficheros" %(lang,len(files[lang])))
    return files
```

Tras ello, se procede a la obtención del conjunto de entrenamiento, así como, el etiquetado para poder realizar el proceso de entrenamiento:

```
X_train, y_train = parse_files(args.dataset)

def get_chunks(words):
    sentences = list()
```

```
    offset = 0
    chunk_len = random.randint(1,20)
    rest = len(words) - chunk_len

    while rest > 0:
        chunk = words[offset:offset+chunk_len]
        sentences.append(' '.join(chunk))
        offset += chunk_len
        chunk_len = random.randint(1,20)
        rest -= chunk_len

    return sentences

def get_sentences(text):
    words = re.compile('\w+').findall(text)
    return get_chunks(words)

def parse_files(data_path):
    X_set = list()
    y_set = list()
    languages = [d for d in listdir(data_path) if not d.startswith('.')]
    for lang in languages:
        files = [f for f in listdir('%s/%s/'%(data_path,lang)) if not
f.startswith('.')]
        for fil in tqdm(files):
            with open('%s/%s/%s'%(data_path, lang, fil)) as f:
                sentences = get_sentences(f.read())
                for sentence in sentences:
                    X_set.append(sentence)
                    y_set.append(lang)
    return X_set,y_set
```

Como se puede observar, se realiza una separación aleatoria del texto en frases de entre 1 y 20 palabras (*get_chunks*). Aunque esto no es obligatorio aquí (sólo en classify.py) se ha unificado en ambos scripts. Si se desea evitar, simplemente se debe sustituir la función *get_sentences* por:

```
def get_sentences(text):
    words = re.compile('\w+').findall(text)
    return ' '.join(words)
```

La expresión regular **\w+** permite obtener un texto con sólo palabras, evitando otros caracteres.

A continuación, se realiza el entrenamiento definiendo un *pipeline* que nos ayuda a procesar la información de forma óptima, y a llevar la salida del *vetorizer* al *classifier*:

```
pipe = Pipeline([
        ('vectorizer', TfidfVectorizer(stop_words=stopwords.words('english'),
min_df=0)),
        ('classifier', MultinomialNB(alpha=0.0005)),
    ])

    model = pipe.fit(X_train,y_train)
```

En este caso, se ha optado por usar la librería sklearn, en concreto **TfidfVectorizer** como utilidad para transformar la información desde europarl a valores numéricos que puede entender la red bayesiana. Exactamente ayuda a extraer la característica de frecuencia de término, es decir, es una medida numérica que expresa cuán relevante es una palabra para un documento en una colección.

Y por otro lado, se ha optado por **MultinomialNB** o **Naive Baiyes MultinomialNB**, por su simpleza y ser adecuada para problemas de clasificación. Más información en el siguiente link.

Por último, se procede a guardar en fichero el modelo entrenado para su posterior uso en classify.py, y cuyo código se muestra a continuación:

```
with open('model.pickle', 'wb') as f:
    pickle.dump(pipe, f, protocol=2)
```

classify.py

Este script es el responsable de realizar la evaluación. Requiere de un corpus de entrada, que debe ser distinto al de entrenamiento, y produce como salida, por consola, una matriz de confusión, como se muestra a continuación:

```
           |                        p
|
           |                        o
|
           |                        r
|
           |            s      e      t      i                 f      s
|
           |      f      p      n      u      t      g          i      w      d
|
           |      r      a      g      g      a      e      d      n      e      a
g |
           |      e      n      l      u      l      r      u      n      d      n
r |
           |      n      i      i      e      i      m      t      i      i      i
e |
           |      c      s      s      s      a      a      c      s      s      s
e |
           |      h      h      h      e      n      n      h      h      h      h
k |
-----------+-------------------------------------------------------------------
-------+
    french | <14.7%>  0.5%   0.3%   0.5%   0.1%   0.2%   0.1%   0.3%   0.3%   0.1%
25.0% |
   spanish |   0.3% <12.0%>  0.1%   0.4%   0.1%   0.0%   0.1%   0.0%   0.0%   0.0%
0.0% |
   english |   0.5%   0.2%  <8.0%>  0.1%   0.1%   0.1%   0.1%   0.1%   0.1%   0.0%
0.7% |
portuguese |   0.1%   0.3%   0.0%  <6.8%>  0.0%   0.0%   0.0%   0.0%   0.0%   0.0%
```

```
0.1% |
   italian |   0.3%   0.2%   0.0%   0.1%  <6.4%>  0.0%   0.0%   0.0%   0.0%   0.0%
0.0% |
    german |   0.1%   0.0%   0.0%   0.0%   0.0%  <6.5%>  0.0%   0.0%   0.0%   0.0%
0.0% |
     dutch |   0.2%   0.1%   0.0%   0.1%   0.0%   0.0%  <4.0%>  0.0%   0.0%   0.0%
0.0% |
   finnish |   0.0%   0.0%   0.0%   0.0%   0.0%   0.0%   0.0%  <3.4%>  0.0%   0.0%
0.0% |
   swedish |   0.0%   0.0%   0.0%   0.0%   0.0%   0.0%   0.0%   0.0%  <3.2%>  0.0%
0.0% |
    danish |   0.1%   0.1%   0.1%   0.1%   0.0%   0.1%   0.0%   0.0%   0.2%  <1.2%>
0.0% |
     greek |      .   0.0%   0.0%      .      .   0.0%      .   0.0%   0.0%      .
<0.0%>|
-----------+-------------------------------------------------------------------------
-------+
```

Esta salida muestra el porcentaje de colisiones que se producen al realizar la evaluación, es decir, qué error comete el clasificador o evaluador.

La ejecución de este script se realiza de la siguiente forma:

```
~$ python3 classify.py model.pickle languages.txt ./wikipedia/
```

En esta ocasión, necesita de: * model.pickle fichero que contiene el modelo generado anteriormente mediante train.py. Un modelo previamente entrenado se puede encontrar aquí. * languages.txt fichero que contiene la lista de lenguajes que se desea evaluar, al igual que en train.py. * ./wikipedia/ es el directorio utilizado para probar el sistema durante su desarrollo. El directorio puede ser cualquier otro con directorios por cada lenguaje en su interior. En este caso, se ha utilizado el subproyecto indicado anteriormente para utilizar wikipedia como corpus y realizar así la evaluación.

En cuanto a la implementación, se vuelve a utilizar argparse, indicando en este caso los parámetros propios de este script:

```
parser = argparse.ArgumentParser(description='Files classification')
parser.add_argument('model', metavar='M', type=str, help='Pickle file')
parser.add_argument('languages', metavar='M', type=str, help='Languages for
training')
parser.add_argument('dataset', metavar='D', type=str, help='Folder containing the
documents within the subfolders')
args = parser.parse_args()
```

Lo siguiente es realizar, igual que en el script train.py, la obtención de la configuración de los idiomas que se desean evaluar, y el directorio con el corpus de evaluación. El corpus debe estár organizado de la misma forma que se organiza el contenido en *europarl_raw/* y que ha quedado descrito previamente:

```
languages_for_classify = list(open(args.languages,'r'))[0].rstrip().split(',')
#Remove \n

files = find_files(args.dataset, languages_for_classify)

to_predict,test = parse_files(args.dataset)
```

Por ello, se omite la explicación, ya que se puede ver en train.py.

En este caso, se emplea la red bayesiana junto con el modelo obtenido por parámetro (fichero *model.pickle*) para hacer la clasificación:

```
predicted = classify(to_predict)

def classify(to_predict):
    with open(args.model, 'rb') as f:
        model = pickle.load(f)
    return model.predict(to_predict)
```

El propio objeto *model* contiene lo necesario para hacer la predicción, es decir, la red bayesiana, la utilidad que vectoriza la entrada, y por último, se utiliza el conjunto 'to_predict' que contiene las frases de entre 1 y 20 palabras a partir del corpus proveniente de wikipedia.

Finalmente, se procede a crear la matriz de confusión, utilizando nltk:

```
predicted_str = to_str(predicted)

confusion_matrix(predicted_str, test)
```

La función *to_str* hace la transformación desde array de NumPy a lista de *String*. La función confusion_matrix utiliza **nltk.ConfusionMatrix** que provee nltk, el cual espera por un lado las etiquetas a partir de la obtención de los ficheros a clasificar, y los obtenidos a partir de la clasificación. Ambas definiciones se presentan a continuación:

```
def to_str(array):
    return [str(item) for item in array]

def confusion_matrix(predicted, test):
    cm = nltk.ConfusionMatrix(predicted, test)
    print(cm.pretty_format(sort_by_count=True, show_percents=True, truncate=11))
```

_classify.py

Este es un script que se puede utilizar de la siguiente forma:

```
~$ python3 classify.py model.pickle "This is an example."
```

Su objetivo es poder ejecutar el clasificador indicando el modelo y el texto a identificar, obteniendo una salida como la siguiente:

```
This is an example. **english**
```

Dado que no entra dentro del objetivo de este trabajo, no se comenta el código, aunque es muy parecido, e incluso más simple, que lo comentado anteriormente.

Tecnologías

Las tecnologías utilizadas son: * python3 * pip3 * scikit-learn * nltk * numpy * scipy * tqdm * pickle * wikipedia