Middle East Technical University | Department of Computer Engineering

CENG 499

Introduction to Machine Learning

Fall '2018-2019
Assignment 1

Yusuf Mücahit Çetinkaya
{yusufc@ceng.metu.edu.tr}
Due date: November 18, 2018, Sunday, 23:59

1 Overview

In this assignment, you are going to predict the genre of a movie by looking its' subtitles. Data is the most vital issue in machine learning and it is just a bulk without preprocessing and extracting features. You need to apply these steps to the given dataset for getting a meaningful representation of each movie. The representations will be fed into a neural network classifier that you will build using Keras API.

Keywords: *movie genre prediction, data preperation, feature extraction, bag of words, tf-idf, neural networks*

2 Dataset

Dataset consists of movie subtitles of 8 genres where each genre has been splitted as 690 train set and 295 test set. Genres are namely **Action, Comedy, Crime, Horror, Musical, Romance, War, Western**. I would like to thank Mesut Gürlek et al. for sharing the dataset that they are collected for their project which is available on GitHub.

3 Tasks

You are expected to develop an end-to-end classification pipeline starting from data preprocessing to training a neural network model. You are given requirements.txt for easy install by creating a virtual environment. You can use the command "pip install -r requirements.txt". If you have installed CUDA dependencies for tensorflow than you can use requirements-gpu.txt.

3.1 Data Preperocessing (15 pts.)

For this task, you will implement the missing methods of **Preprocessor** class where the stub code is already given to you. Preprocessor class simply takes dataset directory and processed dataset directory. preprocess method apply changes to each document under this dataset directory and persists them under the processed dataset directory by keeping the directory structure.
preprocess_document method;

- Tokenizes the document,
- Removes punctuations, words including numbers and words that are in the stop words,
- Stems the words.

You will use nltk for stop words and SnowballStemmer. An example run of preprocess_document is below;

```
from Processor import Processor
text =""" Greetings,
          shall I sit or stand?
          - Tell us.
          - Tell us.
          I'll tell. We bought the goods
          from Black Faik.
          We reloaded the truck
          in Karabuk.
          I was driving the truck
          till Adana.
          - What are you talking about?
          - And you?!
          You've abducted me,
          you'll do the talking.
          I'm confused anyway.
          - Aggressive.
          - Aggressive.
          Yeah, aggressive.
          Is that it?"""
p=Preprocessor()
print p.preprocess_document(text)
# greet shall sit stand tell us tell us tell bought good black faik reload truck ↩
                                                karabuk drive truck till adana talk abduct↩
                                                 talk confus anyway aggress aggress yeah ↩
                                                aggress
```

3.2 Feature Extraction (25 pts.)

You are expected to implement the missing methods of two classes; **OneHotEncoder** and **Vectorizer**.

OneHotEncoder will take list of labels and convert them to one-hot vectors of numpy arrays. It will store the encoding and use it for decoding. Sample run of OneHotEncoder is below;

```
from OneHotEncoder import OneHotEncoder
o=OneHotEncoder()
train_labels=["Action","Comedy","Crime","Comedy","Crime","Musical","Action"]
test_labels=["Comedy","Action","Crime","Musical","Crime","War"]
train_one_hot_vectors=o.fit_transform(train_labels)
test_one_hot_vectors=o.transform(test_labels)
print o.get_feature_names()
# ['Action', 'Comedy', 'Musical', 'Crime']
print train_one_hot_vectors
# [[1 0 0 0]
# [0 1 0 0]
# [0 0 0 1]
# [0 1 0 0]
# [0 0 0 1]
# [0 0 1 0]
# [1 0 0 0]]
print test_one_hot_vectors
#[[0 1 0 0]
# [1 0 0 0]
# [0 0 0 1]
# [0 0 1 0]
# [0 0 0 1]
# [0 0 0 0]]
one_hot_vector=np.array([0,1,0,0])
print o.decode(one_hot_vector)
# Comedy
```

Vectorizer takes min_word_length, max_df, min_df parameters. It only takes terms longer than min_word_length and having document frequency between max_df, min_df into account.

In fit method, it computes document frequencies for each term and generates vocabulary. In transform method, it vectorizes the document with 3 different methods; existance, count, tf-idf. Vector is an array where each index represents a term in vocabulary. Elements in the vector are computed for chosen method as follows;

- Existance($term$, $document$) $= \begin{cases} 1, & \text{if } term \text{ exists in } document \\ 0, & \text{else} \end{cases}$
- Count($term$, $document$) = number of occurences of $term$ in $document$
- TF-IDF($term$, $document$) = TF($term$, $document$)*IDF($term$)

  TF($term$, $document$) = Number of times $term$ appears in $document$

  $$\text{IDF}(term) = log_e\left(\frac{1 + \text{Total number of documents}}{1 + \text{Total number of documents where } term \text{ contained}}\right) + 1$$

  Note: After calculating the feature vector with tf-idf, normalize it with L2 (Euclidean) norm.

An example of Vectorizer run is as follows;

```
from Vectorizer import Vectorizer
v=Vectorizer(min_df=.25, max_df=.75)
contents =[
 "this is the first document",
 "this document is the second document",
 "and this is the third one",
 "is this the first document",
]
v.fit(contents)
print v.get_feature_names()
# ['and', 'third', 'one', 'second', 'document', 'first']
existance_vector =v.transform(contents, method="existance")
print existance_vector
#[[0 0 0 0 1 1]
# [0 0 0 1 1 0]
# [1 1 1 0 0 0]
# [0 0 0 0 1 1]]
count_vector =v.transform(contents, method="count")
print count_vector
#[[0 0 0 0 1 1]
# [0 0 0 1 2 0]
# [1 1 1 0 0 0]
# [0 0 0 0 1 1]]
tf_idf_vector =v.transform(contents, method="tf-idf")
print tf_idf_vector
#[[0. 0. 0. 0. 0.62922751 0.77722116]
# [0. 0. 0. 0.61666846 0.78722298 0. ]
# [0.57735027 0.57735027 0.57735027 0. 0. 0. ]
# [0. 0. 0. 0. 0.62922751 0.77722116]]
```

3.3 Neural Networks (40 pts.)

If you have finished previous tasks, you are ready to train a model to predict the genre of a movie. For this task, you will use Keras API under tensorflow.

You will use one hidden layer with 512 nodes, categorical_hinge loss function, relu activation function in hidden layer, 75 epochs as default model. At output layer, you will always use softmax as activation. Do the followings as your experiment and paste the output of train_model method for each turn into a single file with the title of its' configurations;

1. Fill the missing parts of train_evaluate_model method. Do not forget to set verbose parameter 0 while generating output for the report.

2. Vectorize the preprocessed dataset with min_df values of (0.5, 0.25, 0.1) and max_df value of 0.97 with all of the feature extraction methods(existance, count, tf-idf). Do not forget to fit your vectorizer only with train dataset. Train the model with default configurations and evaluate with given metrics. Choose and continue next one with the best feature and min_df according to top-3-category-accuracy. (Total of 9 runs)

3. Train the model with 3 different hidden layer configurations; [(512,),(512,256),(512,256,128)].

Choose and continue next one with the best hidden layer configuration according to top-3-category-accuracy.(Total of 3 runs)

4. Try sigmoid, relu and tanh activation functions in hidden layers with categorical_crossentropy and categorical_hinge loss functions. Choose and continue next one with the best activation and loss functions according to top-3-category-accuracy.(Total of 6 runs)
5. Run the model for 500 epochs with all chosen best configurations.

3.4 Brain Storming (20 pts.)

Please answer the following questions according to your experiment results in a PDF file under Documents directory;

1. What if we were very lucky to get these accuracy results depending on training and test set distribution? What is the correct way to measure the accuracy?
2. Why did we use top_2 and top_3 accuracy and they are significantly higher than categorical accuracy? Comment on which genres usually misclassified with each other by looking confusion matrices?
3. What are the disadvantages of using bag-of-words approach?
4. What makes the difference between tf-idf and count? What is the importance of idf?
5. What was the effect of changing min_df?

4 Submission & Evaluation

1. You will submit main.py, Preprocessor.py, OneHotEncoder.py, Vectorizer.py and ExperimentSuite.py,
2. You will have **Source** and **Docs** directories. All of your .py files will be in Source directory. Others will be in the Docs directory.If your directory structure is wrong, you will get **penalty**. If you have done something for bonus, explicitly write it in a readme file under Source directory.
3. Your source codes will be tried with different configurations and various inputs.
4. We have zero tolerance policy for **cheating**. People involved in cheating will be punished according to the university regulations and will get 0. You can discuss algorithmic choices, but sharing code between each other or using third party code is **strictly forbidden**. Even if you take a "part" of the code from somewhere/somebody else - this is also cheating. Please be aware that there are "very advanced tools" that detect if two codes are similar. So please do not think you can get away with by changing a code obtained from another source.
5. Do not put datasets, .pyc files, IDE related files etc. Only .py files and documents will be submitted.
6. Zip all directories and name it as <Student_ID> _ <FullNameSurname>.zip and submit it through COW. For example:
e1234567_YusufMucahitCetinkaya.zip