Alexandria University
Faculty of Engineering
Computer and Communication Engineering Programs
CC482: Artificial Intelligence
Fall 2019

SemEval 2020 - Task 6

Monique Ehab 4928
Hana Hesham 4952
Aya Kandil 4699

Table of Contents

1. Introduction

Text classification is the task of assigning a set of predefined categories (in our example definition/no definition) to free-text.
In machine learning, classification is based on past observations.
Using an input of pre-labeled examples as training data, a machine learning algorithm can learn/analyze the different associations between pieces of text in a fast, cost-effective way.
We used some machine language text specifier specified for text analysis, some of which are :

1) **Naive Bayes**
   Based on Bayes's Theorem, it compute the conditional probabilities of occurrence of two events based on the probabilities of occurrence of each individual event.
   In other words : compute the likelihood of a text's belonging to a category

2) **KNN (K-Nearest Neighbor)**
   It is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure.
   A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function

3) **DT (Decision Tree)**
   The goal is to create a training model to predict the class of the target variable by learning simple decision rules inferred from prior data(training data).

4) **Logistic regression**
   It is a statistical model used for two-class classification

2. Problem Statement

- **Subtask 1: Sentence Classification**
  Given a sentence, classify whether or not it contains a definition. This is the traditional definition extraction task.

- **Subtask 2: Sequence Labeling**
  Label each token with BIO tags according to the corpus' tag specification.

- **Subtask 3: Relation Classification**
  Given the tag sequence labels, label the relations between each tag according to the corpus' relation specification

We worked on subtask 1, where we classify our text (sentences) into one of the two classes : has definition , has no definition
Based on our dataset [SENTENCE] [BIN_TAG]
Where the binary tag is 1 if the sentence contains a definition and 0 if the sentence does not contain a definition

3. Approach

The Training Data set is a collection of enriched labeled data to help us train our models, the more data we have, the sharper the model accuracy. A training data set was provided to us by the SemEval-2020 website, it is all in English , all of them are paragraphs in random topics from random sources.

The data is in the form of **columns**
[TOKEN] [SOURCE] [START CHAR] [END CHAR] [TAG] [TAG ID] [ROOT ID] [RELATION]

Where :

- **SOURCE** is the source .txt file of the excerpt
- **START CHAR/END CHAR** are char index boundaries of the token TAG is the label of the token (O if not a B-[TAG] or I-[TAG])
- **TAG ID** is the ID associated with this TAG (0 if none)
- **ROOT ID** is the ID associated with the root of this relation (-1 if no relation/O tag, 0 if root, and TAG_ID of root if not the root)
- **RELATION** is the relation tag of the token (0 if none).

Each line represents a token and its features.

- A single blank line indicates a sentence break;
- Two blank lines indicates a new 3-sentence context window.
- All context windows begin with a sentence id followed by a period.
- These are treated as tokens in the data

3.1 Data preparation

We were provided by the task1_converter.py class. The class has two functions:

1) convert():
    - loop on each file in the directory
    - check if the filename ends with '.deft' extension,
    - If yes call write_converted and pass to it the source path of the file and output path to write converted file
2) write_converted():

    At first we have the lines written in each '.deft' file
    - we loop on all lines to check the whitespaces
    - split each line to get the tokens separately
    - concatenate the tokens to make a sentence
    - stop at the end of each sentence according to the whitespaces
    - add the label given (check the BIO tag to know if the sentence has a definition or not which exist in line_parts[4] starting from 3rd character)
    - Add each sentence with its label to sentences which is a numpy array that is formed of [Sentence][Label]
    - At the end of the file write file to the output path

Now we have the data in our output folder, where each file is readable and labeled if has a definition or not, we make further preparation in the class **divide_def_nodef** :

- Given the converted files we loop on each file and split it into two files one for sentences with definitions and the other without definition and put the file with definitions in positive folder and the one with no definition in negative folder
- We do this with our training data and our testing data
- As we perform these steps we count the number of positive sentences placed in the positive folder and we do the same with the negative sentences as a part of our data analysis

The previous steps are done by splitting each sentence by "\t" and checking the assigned label if 0, it is considered to have no definition

3.2 Data statistics

- Printing the number of splitted files
- Printing the number of sentences with definitions
- Printing the number of sentences with no definitions
- Printing the size of our vocabulary

3.3 Pre-processing

- We use load_files which read each file in two sub-folder, in our case positive and negative
  (The load_files function automatically divides the dataset into data and target sets, as well as it treats each folder inside our main folder as one category and all the documents inside that folder will be assigned its corresponding category)

- For each file, cleaning take place, which is removing numbers, special characters, multiple whitespace as well as turning all the sentences into lowercase sentences and lematizationate words of the sentences.

3.4 Vectorization

- Convert text into the corresponding numerical form
  First we use countVectorizer to be able to save our vocabulary to be later used by our test data
  Then convert it to word embedding model TFIDF

Further explanation

1) **CountVectorizer class from the sklearn.feature_extraction.text library.**
   - The fit_transform function of the CountVectorizer class converts text documents into corresponding numeric features.

2) **Finding TFIDF (TF stands for "Term Frequency",**
IDF stands for "Inverse Document Frequency")

- The bag of words approach works fine for converting text to numbers. However, it has one drawback. It assigns a score to a word based on its occurrence in a particular document.
- It doesn't take into account the fact that the word might also be having a high frequency of occurrence in other documents as well.
- TFIDF resolves this issue by multiplying the term frequency of a word by the inverse document frequency.

The TF is calculated as:
Term frequency = (Number of Occurrences of a word)/(Total words in the document)
IDF(word) = Log((Total number of documents)/(Number of documents containing the word))

- We use different classifiers and GridSearchCV which help us in tuning the parameters (choose best estimator)
  of each classifier through cross-validation
  The GridSearchCV takes the classifier as a parameter and some other parameters in order to help in getting a better model and avoid overfitting (by making it to precise) and underfitting (by making it too general)

  Parameters :
  - Cv: Determines the cross-validation splitting strategy
  - N_jobs : Number of jobs to run in parallel

3.5 Classifiers

1. **Logistic regression :**

   Parameters :

   - C: a regularization parameter which controls the inverse of the regularization strength, a large C can lead to an overfit model, while a small C can lead to an underfit model.
   - multiclass='ovr' : for binary problem
   - solver='liblinear' : since that text classification is nearly linear so we use liblinear that goes with penalty 'L1'

2. **Decision Tree:**

   Parameters:

   - Criterion: (gini,entropy) "gini" for the Gini impurity and "entropy" for the information gain.
   - Max_depth :The maximum depth of the tree.
   - Min_samples_split : The minimum number of samples required to split an internal node

3. **K-Nearest Neighbor:**

   Parameters:

   - N_neighbors int : Number of neighbors to use by default for kneighbors queries.
   - algorithm{'brute'} : 'brute' will use a brute-force search
   - Weights: (uniform, distance)
   - 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
     'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

4. **Naive Bayes:**

Parameters:

- Alpha :: Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).

After we perform each of the classifications we calculate the accuracy of the training data.

At the end of this phase we save our vocabulary and our models to be used in prediction of our test data.

4. Results

Prediction class :

- We read our test data and clean it
- We use the countvectorizer in order to load our saved vocabulary, transform it to tf-idf and then fit our test data to it
- We then use test data as an input for our models
- The output predicted label is then compared with the actual label
- At last we calculate the accuracy, precision, recall and f1-score for each of the models

5. Performance :

Name	Test Accuracy	f1 Measure	recall	precision
NB	0.797	0.796	0.797	0.748
KNN	0.696	0.689	0.696	0.652
DT	0.659	0.646	0.659	0.598
LOG	0.826	0.826	0.826	0.780

6.Conclusion :

- Since the data is text, it is closer to being linearly separable

- Therefore, logistic regression with 'liblinear' solver is the highest

- Naive Bayes is a linear classifier so it gives second best results.
(Naive Bayes is highly accurate when applied to big data)

- K-NN is not linear; k-nn is usually slower for large amounts of data, because of the calculations required for each new step in the process. If speed is important, choose Naive Bayes over K-NN.
(KNN has problems with the curse of dimensionality and large feature sets)

-Decision trees come last
(Due to over-fitting, which is a major problem with decision trees, the issue could (at least, in theory) be avoided by using boosted trees or random forests. In many situations, boosting or random forests can result in trees outperforming either Bayes or K-NN.)