DETAILED DESCRIPTION FOR THE FIRST REVIEW OF THE PROJET

Batch Id : 6B

Project Title: Sentiment Analysis for movie reviews

Project Domain: Machine Learning

Sentiment analysis is a fundamental problem aiming to give a machine the ability to understand the emotions and opinions expressed in a written text. This is an extremely challenging task due to the complexity and variety of human language. The sentiment polarity classification task of EvalItalia-2016 (sentipolc) consists of three subtasks which cover different aspects of sentiment detection: *T*1 : Subjectivity detection: is the tweet subjective or objective? *T*2 : Polarity detection: is the sentiment of the tweet neutral, positive, negative or mixed?

Twitter sentiment classification have intensively researched in recent years .Different approaches were developed for Twitter sentiment classification by using machine learning such as Support Vector Machine (SVM) with rule-based features and the combination of SVMs and Naive Bayes (NB) . In addition, hybrid approaches combining lexicon-based and machine learning methods also achieved high performance described in . However, a problem of traditional machine learning is how to define a feature extractor for a specific domain in order to extract important features.

Deep learning models are different from traditional machine learning methods in that a deep learning model does not depend on feature extractors because features are extracted during training progress. The use of deep learning methods becomes to achieve remarkable results for sentiment analysis . Some researchers used Convolutional Neural Network (CNN) for sentiment classification. CNN models have been shown to be effective for NLP. For example, proposed various kinds of CNN to learn sentiment-bearing sentence vectors, adopted two CNNs in character-level to sentence-level representation for sentiment analysis.constructs experiments on a characterlevel CNN for several large-scale datasets. In addition, Long Short-Term Memory (LSTM) is another state-of-theart semantic composition model for sentiment classification with

many variants described in . The studies reveal that using a CNN is useful in extracting information and finding feature detectors from texts. In addition, a LSTM can be good in maintaining word order and the context of words. However, in some important aspects, the use of CNN or LSTM separately may not capture enough information.

HALF IMPLMENTATION

1) Tweets are firstly considered via a processor based on preprocessing steps and the semantic rules based method in order to standardize tweets and capture only important information containing the main sentiment of a tweet.
2) We use DeepCNN with Wide convolution for character-level embeddings. A wide convolution can learn to recognize specific *n-grams* at every position in a word that allows features to be extracted independently of these positions in the word. These features maintain the order and relative positions of characters. A DeepCNN is constructed by two wide convolution layers and the need of multiple wide convolution layers is widely accepted that a model constructing by multiple processing layers have the ability to learn representations of data with higher levels of abstraction . Therefore, we use DeepCNN for character level embeddings to support morphological and shape information for a word. The Deep CNN produces *N* global fixed-sized feature vectors for *N* words.
3) A combination of the global fixed-size feature vectors and word-level embedding is fed into Bi-LSTM. The Bi-LSTM produces a sentence-level representation by maintaining the order of words.

CODE TILL FIRST REVIEW

```
# Saving the classifier


import nltk
import random
from nltk.corpus import movie_reviews
import pickle
import sklearn?

nltk.download('stopwords')
nltk.download('movie_reviews')
# this takes the most of the algorithm time.
documents = [(list(movie_reviews.words(fileid)), category)
        for category in movie_reviews.categories()
```

```
            for fileid in movie_reviews.fileids(category)]

    random.shuffle(documents)

    all_words = []
    for w in movie_reviews.words():
        all_words.append(w.lower())

    all_words = nltk.FreqDist(all_words)

    word_features = list(all_words.keys())[:3000]

    def find_features(document):
        words = set(document)
        features = {}
        for w in word_features:
            features[w] = (w in words)

        return features

    #print((find_features(movie_reviews.words('neg/cv000_29416.txt'))))

    featuresets = [(find_features(rev), category) for (rev, category) in documents]

# Training and testing sets splitted up.

    training_set = featuresets[:1900]
    testing_set = featuresets[1900:]

    # posterior = prior occurences * likelihoood / evidence
    classifier = nltk.NaiveBayesClassifier.train(training_set)

    #if you want to save and load in between.
    #classifier_f = open("naivebayes.pickle", "rb")
    #classifier = pickle.load(classifier_f)
    #classifier_f.close()

# Testing now.
    print("Naive Bayes Algorithm accuracy percent:", (nltk.classify.accuracy(classifier, testing_set))*100)

    # most valuable words when it comes to positive and negative movie reviews.
    classifier.show_most_informative_features(15)

    # saving the classifier
    save_classifier = open("naivebayes.pickle", "wb")
    pickle.dump(classifier, save_classifier)
    save_classifier.close()
```

Project guide signature