Pre-processing Documentation

Objective:

Text **pre-processing** is an important step for natural language processing (**NLP**) tasks. It transforms **text** into a more digestible form so that machine learning algorithms can perform better.

Required Packages:

**nltk:** (Natural Language Toolkit) NLTK is a platform for building Python programs to work with human language data. It is required for basic pre-processing operations like tokenization, stopwords removal, lemmatization and stemming.

Installation:

In command prompt, enter command:

pip install nltk

In python file, download required modules:

import nltk

nltk.download()

**sklearn:** It is a set of python modules for machine learning and data mining. It is required for calculating tf-idf, applying PCA, variance threshold and clustering algorithms.

Installation:

In command prompt, enter command:

pip install scikit-learn

**BeautifulSoup:** Beautiful Soup is a library that makes it easy to scrape information from web pages. It is required for scraping data from termsheet links.

Installation:

In command prompt, enter command:

pip install beautifulsoup4

**pandas:** Itis a python library for data manipulation and analysis. It is required for data structures like dataframes, vectors and their manipulation.

Installation:

In command prompt, enter command:

pip install pandas

Data cleaning steps :

1. Convert all text in ASCII characters
2. Remove newline characters
3. Convert all letters to lowercase
4. Remove URLs
5. Remove numbers
6. Remove punctuations and unknown characters
7. Remove unusual words (words having no meaning)
8. Remove stopwords
9. Lemmatization
10. Stemming
11. Remove the most common(appearing in more than 75% of documents) and rare words (appearing in only one document) (though handled by tf-idf)

def preprocessing(textdata, steps):

    '''

    Main function which performs all the pre-processing techniques that the user has selected.

    The techniques to be applied are present in the "steps" list.

    '''

    data = []

    count = 0

    steps = set(steps)

    for text in textdata:

        # will be able to process all characters

        text = unidecode(text)

        text = rmv\_newline\_char(text)

        text = text.lower()

        #Checking in steps to customize preprocessing tasks

        if 'url' in steps:

            text = rmv\_URLs(text)

        text = rmv\_numbers(text)

        text = rmv\_punct(text)

        text = rmv\_unknown\_char(text)

        if 'unusual' in steps:

            text = unusual\_words(text)

        if 'stopwords' in steps:

            text = rmv\_stopWords(text)

        if 'lemmatization' in steps:

            text = apply\_lemmatization(text)

        if 'stemming' in steps:

            text = apply\_stemming(text)

        data.append(text)

    DF = get\_count(data)        #Document frequency of each word

    data = rmv\_common\_words(data, DF)

    return data

This function returns cleaned data.

Usage:

    steps = ['url', 'stopwords', 'lemmatization', 'stopwords']

    ISINs, URLs, textlist = extract.extract('ISINS\_v3.xlsx')

    data = clean\_file.preprocessing(textlist, steps)

Dimensionality Reduction:

1. Calculating tf-idf: Calculates the tf-idf matrix for the input pre-processed data

Arguments: text: pre-processed data

Returns: dataframe representing tf-idf matrix

def tfidf(text):

    vectorizer = TfidfVectorizer(smooth\_idf=False)

    vectors = vectorizer.fit\_transform(text)

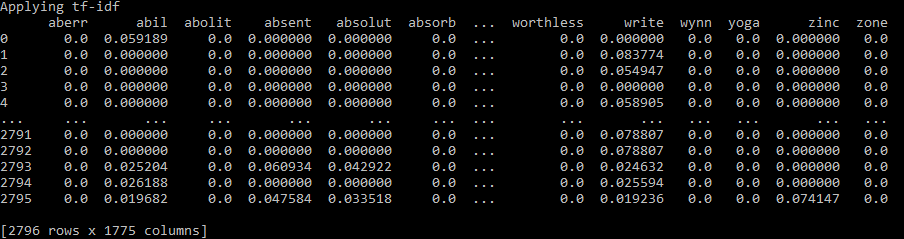
    features = vectorizer.get\_feature\_names()

    dense = vectors.todense()

    denselist = dense.tolist()

    df = pd.DataFrame(denselist, columns=features)

    return df



Using all pre-processing techniques, we got 1775 features.

1. Apply variance threshold: Reduces features in the tf-idf matrix using variance threshold

Arguments: tfidf: tf-idf dataframe

thresh: thresholdof variance

Returns: reduced tf-idf dataframe

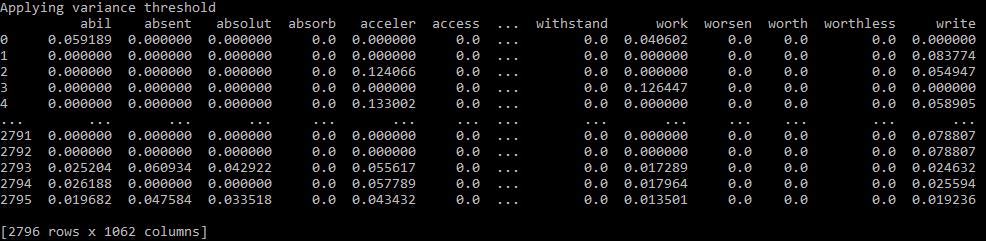
def varThresh\_tfidf(tfidf, thresh):

    selector = VarianceThreshold(threshold=thresh)

    selector.fit(tfidf)

    data = tfidf[tfidf.columns[selector.get\_support(indices=True)]]

    return data



After applying variance threshold of 0.0001, we got 1062 features.

1. Apply PCA : Principal Component Analysis to reduce features

Arguments: tfidf: tf-idf dataframe

n: number of components

Returns: ratio: numpy array representing ratio of reduced data

scores: numpy array representing principal components

df: dataframe of reduced features

def pca\_tfidf(tfidf, n):

    scaler = StandardScaler()

    segmentation\_std = scaler.fit\_transform(tfidf)

    pca = PCA(n\_components=n)

    scores\_pca = pca.fit\_transform(segmentation\_std)

    df = pd.DataFrame(pca.components\_,columns=tfidf.columns)

    ratio = pca.explained\_variance\_ratio\_

    return ratio, scores\_pca, df

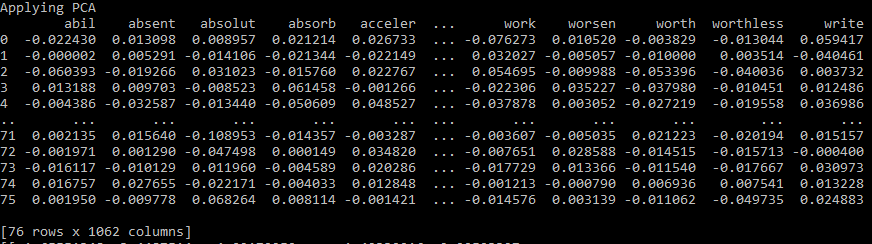
Usage:

    data = clean\_file.preprocessing(textlist, steps)

    df = clean\_file.tfidf(data)

    tfidf = clean\_file.varThresh\_tfidf(df, 0.0001)

    ratio, score, pcadf = clean\_file.pca\_tfidf(tfidf, 0.8)



After applying PCA, the number of features is reduced to 76.