Predict tags for job offers with linear models

In this notebook we are going to predict tags for job offers.

tags: ['company' 'jobdescription' 'joblocation_address' 'jobtitle' 'skills']

Libraries

- Numpy - a package for scientific computing.
- Pandas - a library providing high-performance, easy-to-use data structures and data analysis tools for the Python
- Scikit-learn - a tool for data mining and data analysis.
- NLTK - a platform to work with natural language.

Train and Validation Data

In [9]:

```
from ast import literal_eval
import pandas as pd
import numpy as np
```

In [81]:

```
df = pd.read_csv('df_dropna.csv')
df.head()
```

Out[81]:

	Sentence	Tag
0	development engineer – time metrology	skills
1	whose mandate provide basis coherent system m...	company
2	bipm base sèvres outskirts staff 70	company
3	laboratory equip characterization gnss time t...	skills
4	principal duty responsibility reporting direct...	skills

In [ ]:

```
df.rename(columns = {
    'Sentence': 'text',
    'Tag': 'tag'
}, inplace = True)
df['tag'] = df['tag'].str.replace('comapny', 'company')
df = df.dropna()
```

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(['company', 'jobdescription', 'joblocation_address', 'jobtitle', 'skills'
])
df.tag = le.transform(df.tag)
```

In [84]:

```
df.head()
```

Out[84]:

	text	tag
0	development engineer – time metrology	4
1	whose mandate provide basis coherent system m...	0
2	bipm base sèvres outskirts staff 70	0
3	laboratory equip characterization gnss time t...	4
4	principal duty responsibility reporting direct...	4

Test Data

In [ ]:

```
test = pd.read_csv('test_df.csv', names = ["text"], skiprows=1)
test.dropna(inplace = True)
```

train_test_split

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(df.text, df.tag, random_state
= 0)
```

In [ ]:

```
X_train = X_train.values
X_val = X_val.values
y_train = y_train.values
y_val = y_val.values

X_test = test['text'].values
```

Text preprocessing

In [5]:

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/fairouz/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [6]:

```
def read_data(filename):
    data = pd.read_csv(filename, sep='\t')
    data['tags'] = data['tags'].apply(literal_eval)# drop the '' from inside tag
s
    return data
```

In [3]:

```
import re
```

In [4]:

```
REPLACE_BY_SPACE_RE = re.compile('[/(){}\[\]\|@,;]')
BAD_SYMBOLS_RE = re.compile('[^0-9a-z #+_]')
STOPWORDS = set(stopwords.words('english'))

def text_prepare(text):

    text = text.lower()
    text = re.sub(REPLACE_BY_SPACE_RE, " ", text)
    text = re.sub(BAD_SYMBOLS_RE, "", text)
    text = " ".join([word for word in re.split(" ", text) if (not word in STOPWO
RDS and not word == "")])
    return text
```

Now we preprocess the text using function *text_prepare*:

In [ ]:

```
X_train = [text_prepare(x) for x in X_train]
X_val = [text_prepare(x) for x in X_val]

X_test = [text_prepare(x) for x in X_test]
```

In [93]:

```
X_train[:3]
```

Out[93]:

```
['spirit community diversity people skill background experience make
us strong',
 'working large global organization youll chance grow professionally
personally expand career',
 'different']
```

For each tag and for each word we calculate how many times they occur in the train corpus.

In [94]:

```
# Dictionary of all words from train corpus with their counts.
words_counts = {}

from collections import Counter

print("counting words..")
words = []
for title in X_train:
    words = words + title.split()

print("done")
words_counts = Counter(words)
```

```
counting words..
done
```

In [95]:

```
len(words_counts)
```

Out[95]:

```
5358
```

Bag Of Words

In [ ]:

```
DICT_SIZE = 5000
WORDS_TO_INDEX = {word:i for i,word in enumerate([word for word,_ in sorted(word
s_counts.items(), key=lambda x: x[1], reverse=True)[:DICT_SIZE]])}
INDEX_TO_WORDS = dict(enumerate(list(WORDS_TO_INDEX)))
ALL_WORDS = WORDS_TO_INDEX.keys()

def my_bag_of_words(text, words_to_index, dict_size):

    result_vector = np.zeros(dict_size)
    for word,i in words_to_index.items():
        if word in text.split():
            result_vector[i] = result_vector[i] + 1

    return result_vector
```

Now we apply the implemented function to all samples:

In [ ]:

```
from scipy import sparse as sp_sparse
```

In [98]:

```
#X_train_mybag.toarray() to decompress the matrix
X_train_mybag = sp_sparse.vstack([sp_sparse.csr_matrix(my_bag_of_words(text, WOR
DS_TO_INDEX, DICT_SIZE)) for text in X_train])
X_val_mybag = sp_sparse.vstack([sp_sparse.csr_matrix(my_bag_of_words(text, WORDS
_TO_INDEX, DICT_SIZE)) for text in X_val])

X_test_mybag = sp_sparse.vstack([sp_sparse.csr_matrix(my_bag_of_words(text, WORD
S_TO_INDEX, DICT_SIZE)) for text in X_test])

print('X_train shape ', X_train_mybag.shape)
print('X_val shape ', X_val_mybag.shape)
print('X_test shape ', X_test_mybag.shape)
```

```
X_train shape  (3790, 5000)
X_val shape  (1264, 5000)
X_test shape  (5054, 5000)
```

We transform the data to sparse representation, to store the useful information efficiently.

Sklearn algorithms can work only with **csr** matrix, so we will use this one.

TF-IDF

The second approach extends the bag-of-words framework by taking into account total frequencies of words in the corpora. It helps to penalize too frequent words and provide better features space.

In [ ]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [ ]:

```
def tfidf_features(X_train, X_val, X_test):

    tfidf_vectorizer = TfidfVectorizer(min_df=5, max_df=0.9, ngram_range=(1, 2),
token_pattern= '(\S+)')

    X_train=tfidf_vectorizer.fit_transform(X_train)
    X_val=tfidf_vectorizer.transform(X_val)

    X_test=tfidf_vectorizer.transform(X_test)

    return X_train, X_val, X_test, tfidf_vectorizer.vocabulary_
```

Now we apply the implemented function to all samples:

In [ ]:

```
X_train_tfidf, X_val_tfidf, X_test_tfidf, tfidf_vocab = tfidf_features(X_train,
X_val, X_test)
tfidf_reversed_vocab = {i:word for word,i in tfidf_vocab.items()}
```

Shape Compare

In [102]:

```
print(" Shape of X_train_mybag:", X_train_mybag.shape)
print(" Shape of X_train_tfidf:", X_train_tfidf.shape)
```

```
 Shape of X_train_mybag: (3790, 5000)
 Shape of X_train_tfidf: (3790, 1958)
```

Training

In [ ]:

```
from sklearn.linear_model import LogisticRegression
```

In [ ]:

```
def train_classifier(X_train, y_train):

    clf = LogisticRegression().fit(X_train, y_train)
    return clf
```

Training the classifiers for different data transformations: *bag-of-words* and *tf-idf*.

In [ ]:

```
classifier_mybag = train_classifier(X_train_mybag, y_train)
classifier_tfidf = train_classifier(X_train_tfidf, y_train)
```

Now we create predictions for the data. We will need two types of predictions: labels and scores.

In [ ]:

```
y_val_predicted_labels_mybag = classifier_mybag.predict(X_val_mybag)
y_val_predicted_scores_mybag = classifier_mybag.decision_function(X_val_mybag)

y_val_predicted_labels_tfidf = classifier_tfidf.predict(X_val_tfidf)
y_val_predicted_scores_tfidf = classifier_tfidf.decision_function(X_val_tfidf)
```

Now we can take a look at how classifier, which uses TF-IDF, works for a few examples:

In [107]:

```
for i in range(5):
    print('Text:\t{}\nTrue labels:\t{}\nPredicted labels:\t{}\n\n'.format(
        X_val[i],
        list(le.inverse_transform(y_val))[i],
        list(le.inverse_transform(y_val_predicted_labels_mybag))[i]
    ))
```

```
Text:   40 country 5 continent
True labels:    company
Predicted labels:       skills


Text:   adhere agile project set direction data science initiative q
ualifications masters bachelors quantitative discipline eg
True labels:    skills
Predicted labels:       skills


Text:   parametrize exposure model
True labels:    skills
Predicted labels:       skills


Text:   excellent write verbal communication skill include report wr
ite technical document
True labels:    skills
Predicted labels:       skills


Text:   mathematics etc
True labels:    skills
Predicted labels:       skills
```

Evaluation

To evaluate the results we will use several classification metrics:

- Accuracy
- F1-score

In [7]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
```

In [8]:

```
def print_evaluation_scores(y_val, predicted):

    accuracy=accuracy_score(y_val, predicted)
    f1_score_macro=f1_score(y_val, predicted, average='macro')
    f1_score_micro=f1_score(y_val, predicted, average='micro')
    f1_score_weighted=f1_score(y_val, predicted, average='weighted')

    print("Accuracy:", accuracy)
    print("F1-score macro:", f1_score_macro)
    print("F1-score micro:", f1_score_micro)
    print("F1-score weighted:", f1_score_weighted)
```

In [110]:

```
print('Bag-of-words')
print_evaluation_scores(y_val, y_val_predicted_labels_mybag)
print()
print('Tfidf')
print_evaluation_scores(y_val, y_val_predicted_labels_tfidf)
```

```
Bag-of-words
Accuracy: 0.7468354430379747
F1-score macro: 0.4713936370333386
F1-score micro: 0.7468354430379747
F1-score weighted: 0.7188770060462132

Tfidf
Accuracy: 0.7484177215189873
F1-score macro: 0.35538080894431534
F1-score micro: 0.7484177215189873
F1-score weighted: 0.6965920924203463
```

Predictions for *test* set

In [112]:

```
y_test_predicted_labels_mybag = classifier_mybag.predict(X_test_mybag)
y_test_predicted_labels_tfidf = classifier_tfidf.predict(X_test_tfidf)

result_part1 = pd.DataFrame()
result_part1["text"] = X_test
result_part1["prediction BagOfWords"] = le.inverse_transform(y_test_predicted_la
bels_mybag)
result_part1["prediction TFIDF"] = le.inverse_transform(y_test_predicted_labels_
tfidf)

result_part1.head()
```

Out[112]:

	text	prediction BagOfWords	prediction TFIDF
0	project may deliver hardware system technical ...	skills	skills
1	key responsibility specific responsibility job...	skills	skills
2	help invent prototype technology	skills	skills
3	create haptic interface beyond	skills	skills
4	research contribute emerge technology standard...	skills	skills