iNetworks

Job title Classification by industry

(Multi-text Text Classification Task)

Report

Omneya Essam Kamal

Overview

You can think of the job industry as the category or general field in which you work. On a job application, "industry" refers to a broad category under which a number of job titles can fall. For example, sales is an industry; job titles under this category can include sales associate, sales manager, manufacturing sales rep, pharmaceutical sales and so on.

NLP Problem

The problem is a supervised text classification problem, and our goal is to find the most suitable supervised machine learning model to solve it. Given a job title that comes in, we want to assign it to one of 4 industry categories.

Dataset

The dataset contains of more than 8500 data samples featuring 2 columns:

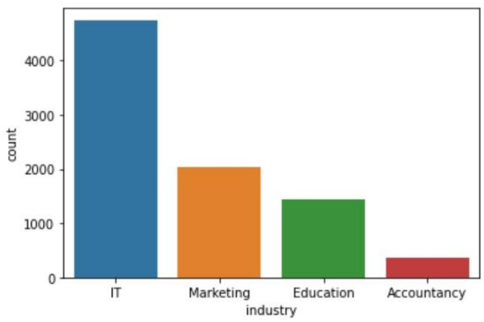
1- job title   2- industry

```
[132] print('Length of the dataset : {} data samples'.format(len(df)))
      print('Dataset columns : {}'.format(df.columns))
```

```
Length of the dataset : 8586 data samples
Dataset columns : Index(['job title', 'industry'], dtype='object')
```

As shown in the below figure, there is class imbalance in the data

```
sns.countplot(data=df, x='industry');
txt="As we can see there is class imbalance here"
plt.figtext(0.5, -0.1, txt, wrap=True, horizontalalignment='center', fontsize=12);
```



As we can see there is class imbalance here

Data cleaning

1- Nulls: data contains no nulls
2- Duplicates: data has more than 4000 duplicates

```
[133] print('Number of duplicates in dataset = {}'.format(df.duplicated().sum()))
```

```
Number of duplicates in dataset = 4618
```

We will follow 2 approaches:

First: not removing duplicates, as this might be changing in real data not duplicated data

Second: removing duplicates, as this may bias the ML model

Text cleaning

```
df['job_title_proc'] = df['job title'].str.lower()
df['job_title_proc'] = df['job_title_proc'].str.replace(r"\(.*\)","")
df['job_title_proc'] = df['job_title_proc'].str.replace(r'\b-\b', " ")
df['job_title_proc'] = df['job_title_proc'].str.replace(r'/', " ")

df['job_title_proc'] = df['job_title_proc'].str.split('-').str[0]
df['job_title_proc'] = df['job_title_proc'].str.split(',').str[0]
df['job_title_proc'] = df['job_title_proc'].str.replace(r'"', "")
df['job_title_proc'] = df['job_title_proc'].str.split('-').str[0]
df['job_title_proc'] =  df['job_title_proc'].apply(lambda x: x.replace(']','').replace('[',''))
df['job_title_proc'] =  df['job_title_proc'].str.replace('\d+', '')
df['job_title_proc'] =  df['job_title_proc'].str.replace('£','')

df['job_title_proc'] =  df['job_title_proc'].str.replace('[{}]'.format(string.punctuation), '')
df['job_title_proc'] =  df['job_title_proc'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stopwords)]))
df['job_title_proc'] =  df['job_title_proc'].str.replace('[^a-zA-Z] ', '')
```

1- Removing stop words
2- Lowering words
3- Removing punctuation
4- Removing strings between parentheses

8017	homeroom teacher (native speaker of english)	Education	homeroom teacher

5- Removing special character and digits
6- Splitting composite job title like the example below

	english instructor (kids-adults) -helwan, maadi	Education	english instructor
7822	senior teacher –young learners- cairo, egypt (egy-s-00034)	Education	senior teacher

7- Lemmatization

Models

Traditional Machine Learning

1- Without any data handling

Just using **CountVectorizer**

Testing with different models:

```
accuracy = []
names = []
for name,model in models:
  model.fit(X_train_dtm, y_train)
  y_pred_class = model.predict(X_test_dtm)
  names.append(name)
  accuracy.append(accuracy_score(y_test, y_pred_class))

for i in range(len(names)):
    print("{} accuracy = {:.3f}".format(names[i],accuracy[i]))
```

```
LogisticRegression accuracy = 0.930
MultinimialNB accuracy = 0.914
SVC accuracy = 0.937
LinearSVC accuracy = 0.932
RandomForest accuracy = 0.923
```

LinearSVC is the one I used in deploying Flask API, because it has the highest accuracy

Although I prefer to use **MultinomialNB** as it's computationally very efficient and easy to implement with text data. Plus, it also fits well with multi-class text classification tasks

2- Handling imbalanced data

1- **SMOTE():** In a classic oversampling technique, the minority data is duplicated from the minority data population. ... SMOTE works by utilizing a k-nearest neighbour algorithm to create synthetic data.

2- **RandomOverSampler():** over-sample the minority class(es) by picking samples at random with replacement.

3- **Class_weight = "balanced"** : is an argument given in the instance of ML model

```
model = make_pipeline_imb(TfidfVectorizer(), SMOTE(), LogisticRegression(class_weight="balanced"))
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy: ", accuracy_score(y_test, y_pred))
print("Confusion matrix: ", confusion_matrix(y_test, y_pred,labels=labels))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function safe_i
  warnings.warn(msg, category=FutureWarning)
Accuracy:  0.9333954354913834
Confusion matrix:  [[1146   27    2   11]
 [  28  469   21   11]
 [  11   14  299   11]
 [   2    4    1   90]]
```

Logistic Regression and LinearSVC have the best results of about 93.5

""

In my opinion: **F1 score** is the best metric suited to this case, followed by accuracy or AUC-ROC, as we do care about both recall and precision and we are not sensitive to any of them

""

3- Removing duplicates

Accuracy has decreased after removing duplicates from model, this is expected as the data size has decreased by its half

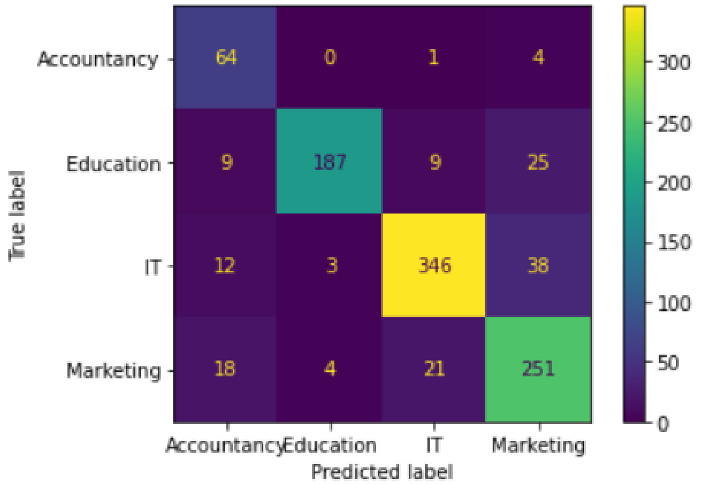
```
[41] model = make_pipeline_imb(TfidfVectorizer(), SMOTE(), LinearSVC(class_weight='balanced'))
     model.fit(X_train, y_train)
     y_pred = model.predict(X_test)
     print("Accuracy: ", accuracy_score(y_test, y_pred))
     print("Confusion matrix: ", confusion_matrix(y_test, y_pred,labels=labels))
```

```
Accuracy:  0.8729838709677419
Confusion matrix:  [[351  37   5   6]
 [ 25 258   4   7]
 [  9  21 195   5]
 [  1   4   2  62]]
```

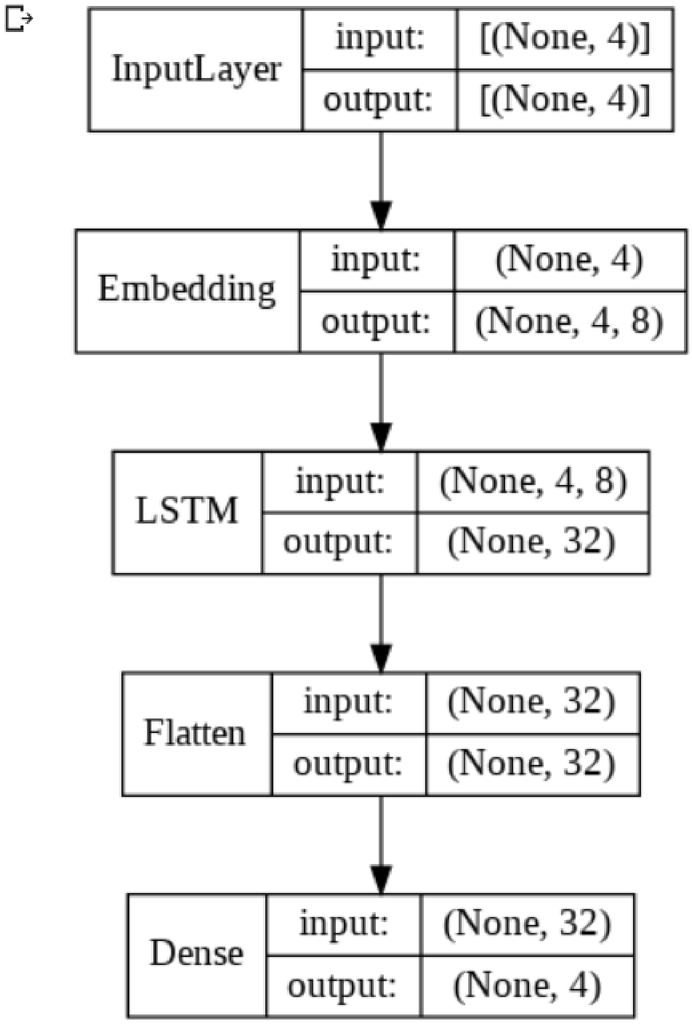
```
model = make_pipeline_imb(TfidfVectorizer(), SMOTE(), MultinomialNB())
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy: ", accuracy_score(y_test, y_pred))
print("Confusion matrix: ", confusion_matrix(y_test, y_pred,labels=labels))

plot_confusion_matrix(model, X_test, y_test,values_format='d')
plt.show()
```

```
Accuracy:  0.8548387096774194
Confusion matrix:  [[346  38   3  12]
 [ 21 251   4  18]
 [  9  25 187   9]
 [  1   4   0  64]]
```



RNN model



- **RNN model is overfitting the data, as the data is very simple and small**

- **How can to extend the model to have better performance?**
  RNN can be restructured to have better performance
  ML models can be tuned with Grid search
- **Limitations of the model**: dataset is very small and the text is too simple as every text sample consists of 2 or 3 words at most

Flask API

- **Request:** " http://127.0.0.1:5000/teacher" (Job title added to request)
- **Response:** "Education" (Predicted Industry by model)