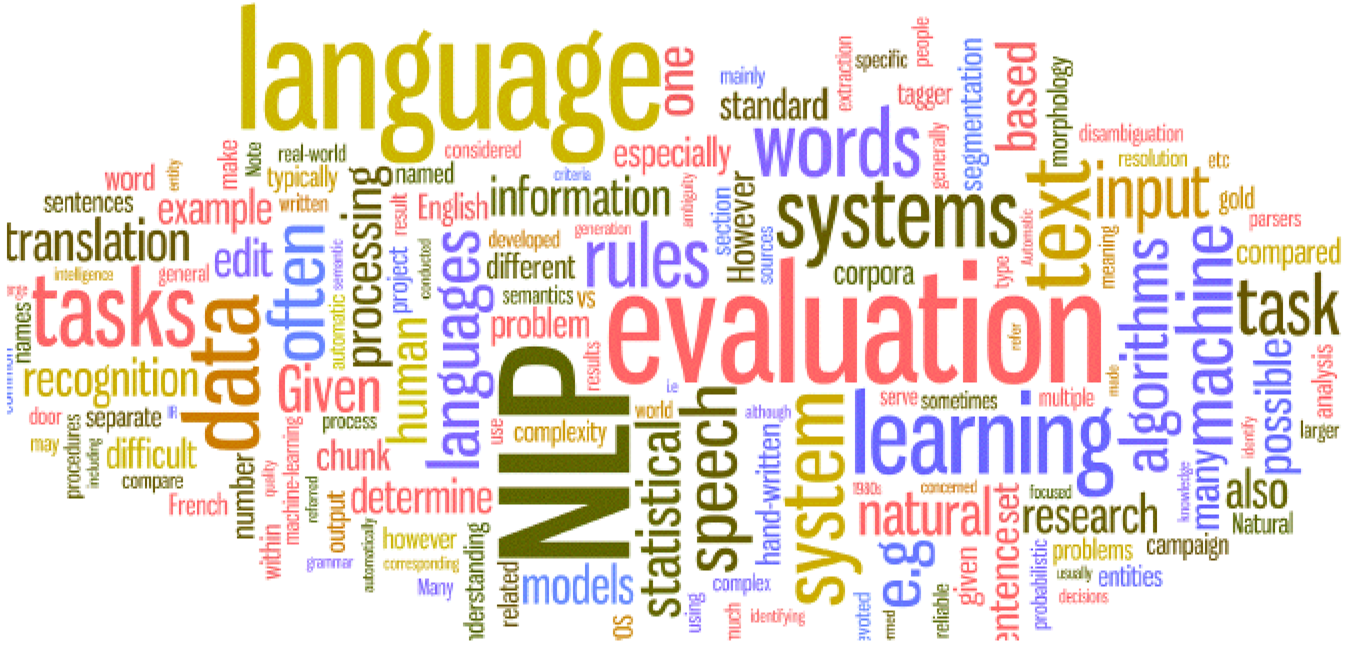


MULTI-TEXT CLASSIFICATION

"Job title Classification by industry"

BY

Adel Atef Adel Ghali Meleka

TABLE OF CONTENTS

PROBLEM STATEMENT

Description

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 several job titles can fall. For example, sales are an industry; job titles under this category can include sales associate, sales manager, manufacturing sales rep, pharmaceutical sales and so on.

Details

You are given a dataset that has two variables (Job title & Industry) in a csv format of more than 8,500 samples.
This dataset is imbalanced (Imbalance means that the number of data points available for different classes is different) as follows:

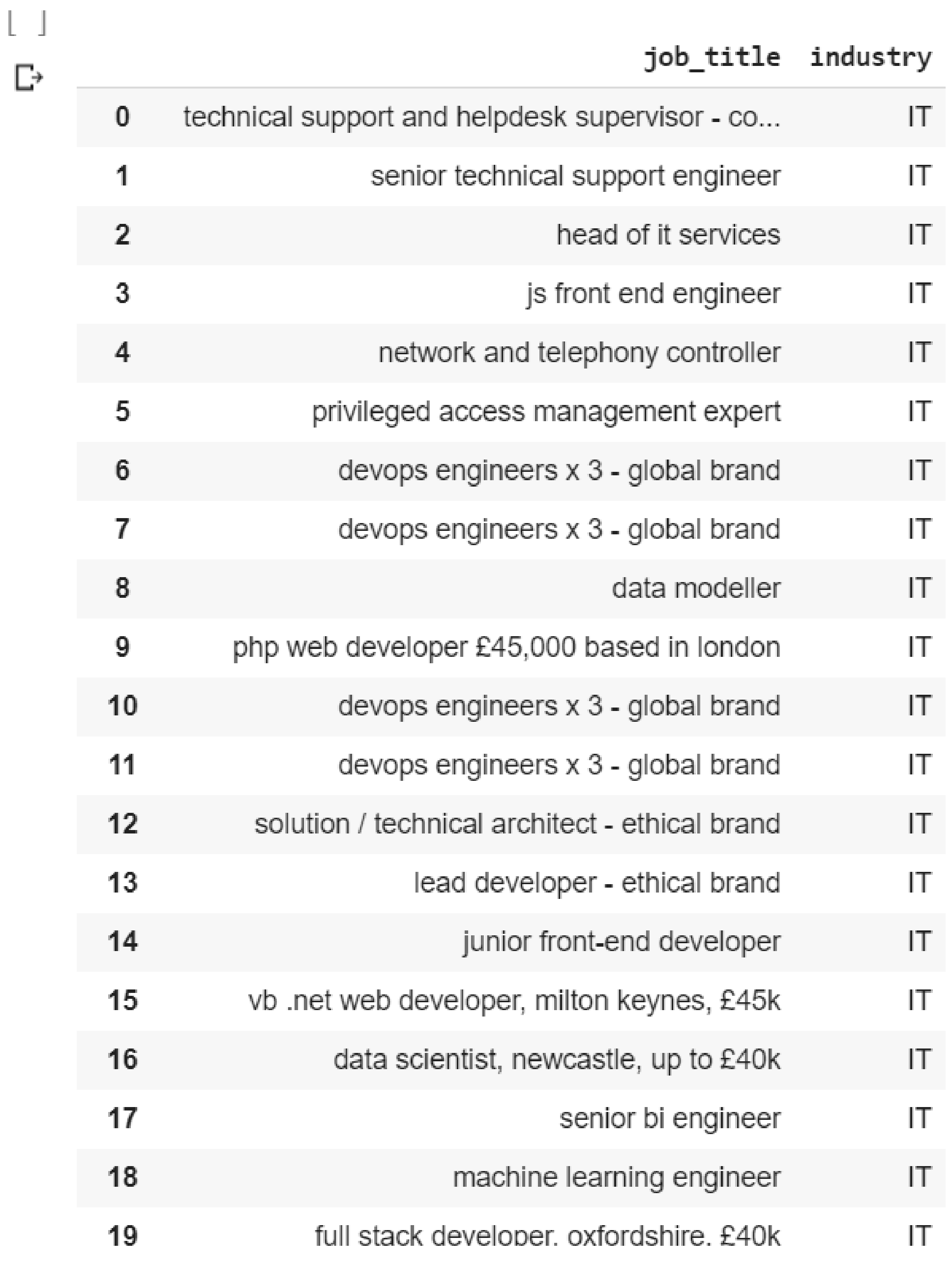
IT	4746
Marketing	2031
Education	1435
Accountancy	374

You are required to build a model using any Machine Learning classifier algorithm to classify job titles by the industry and provide us with insights on how your model works.

DATA LOAD &PREPROCESSING

Data Load from Google Drive

I Uploaded the dataset in my google drive “drive/Collab” directory and linked it into code.
Using Pandas, I created a Data Frame containing our data set as follows:



	job_title	industry
0	technical support and helpdesk supervisor - co...	IT
1	senior technical support engineer	IT
2	head of it services	IT
3	js front end engineer	IT
4	network and telephony controller	IT
5	privileged access management expert	IT
6	devops engineers x 3 - global brand	IT
7	devops engineers x 3 - global brand	IT
8	data modeller	IT
9	php web developer £45,000 based in london	IT
10	devops engineers x 3 - global brand	IT
11	devops engineers x 3 - global brand	IT
12	solution / technical architect - ethical brand	IT
13	lead developer - ethical brand	IT
14	junior front-end developer	IT
15	vb .net web developer, milton keynes, £45k	IT
16	data scientist, newcastle, up to £40k	IT
17	senior bi engineer	IT
18	machine learning engineer	IT
19	full stack developer, oxfordshire, £40k	IT

Dataset Cleaning

I used many Techniques for cleaning the dataset:

Lower Case

I transformed all record lines into lower case letters to avoid double meaning for same word (i.e. Engineer - engineer).

Non-Significant words (Noise Removal)

Any piece of text which is not relevant to the context of the data and the end-output can be specified as the noise.
For example – language stop-words (commonly used words of a language – is, am, the, of, in, etc.), URLs or links, social media entities (mentions, hashtags), punctuations and industry specific words. This step deals with removal of all types of noisy entities present in the text.

A general approach for noise removal is to prepare a dictionary of noisy entities and iterate the text object by tokens (or by words), eliminating those tokens which are present in the noise dictionary.

Instead of preparing a dictionary of noisy entities, we can use a noise list imported from **nltk module.**
In order to exclude noisy words from each tweet line, each line must first be **tokenized.**

Lemmatization

Another type of textual noise is about the multiple representations exhibited by single word.

For example – “play”, “player”, “played”, “plays” and “playing” are the different variations of the word – “play”, Though they mean different but contextually all are similar. The step converts all the disparities of a word into their normalized form (also known as lemma).
The most common lexicon normalization practices are:

Stemming
Stemming is a rudimentary rule-based process of stripping the suffixes (“ing”, “ly”, “es”, “s” etc.) from a word.
There are different algorithms that can be used in the stemming process, but the most common in English is Porter stemmer. The rules contained in this algorithm are divided in five different phases numbered from 1 to 5. The purpose of these rules is to reduce the words to the root.

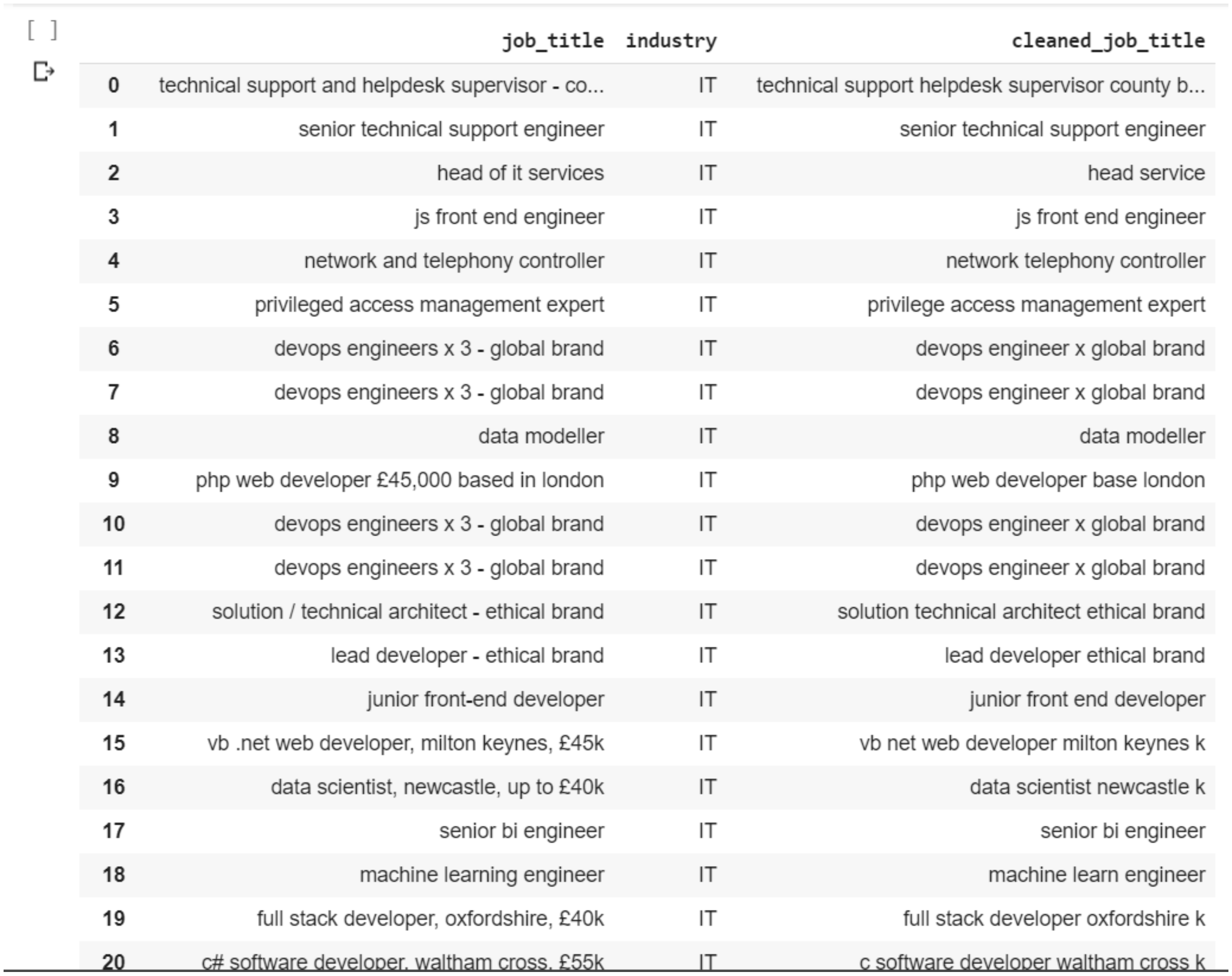
Lemmatization
Lemmatization, on the other hand, is an organized & step by step procedure of obtaining the root form of the word, it makes use of vocabulary (dictionary importance of words) and morphological analysis (word structure and grammar relations).
The key to this methodology is linguistics. To extract the proper lemma, it is necessary to look at the morphological analysis of each word. This requires having dictionaries for every language to provide that kind of analysis.

Lemmatizing is more powerful because it gives a meaningful word.
That’s why I’ve chosen it as our implementation in noise removal.

Special keywords

I also manually prepared a small array containing some other noisy words not removed yet by the whole process in order to completely clean the records.

After successfully making all previous steps, I simply added a new column in my data frame containing a final cleaned version of Job Titles field as follows:



	job_title	industry	cleaned_job_title
0	technical support and helpdesk supervisor - co...	IT	technical support helpdesk supervisor county b...
1	senior technical support engineer	IT	senior technical support engineer
2	head of it services	IT	head service
3	js front end engineer	IT	js front end engineer
4	network and telephony controller	IT	network telephony controller
5	privileged access management expert	IT	privilege access management expert
6	devops engineers x 3 - global brand	IT	devops engineer x global brand
7	devops engineers x 3 - global brand	IT	devops engineer x global brand
8	data modeller	IT	data modeller
9	php web developer £45,000 based in london	IT	php web developer base london
10	devops engineers x 3 - global brand	IT	devops engineer x global brand
11	devops engineers x 3 - global brand	IT	devops engineer x global brand
12	solution / technical architect - ethical brand	IT	solution technical architect ethical brand
13	lead developer - ethical brand	IT	lead developer ethical brand
14	junior front-end developer	IT	junior front end developer
15	vb .net web developer, milton keynes, £45k	IT	vb net web developer milton keynes k
16	data scientist, newcastle, up to £40k	IT	data scientist newcastle k
17	senior bi engineer	IT	senior bi engineer
18	machine learning engineer	IT	machine learn engineer
19	full stack developer, oxfordshire, £40k	IT	full stack developer oxfordshire k
20	c# software developer, waltham cross, £55k	IT	c software developer waltham cross k

CLASSIFICATION MODELS

Feature Engineering on Text

The classifiers and learning algorithms cannot directly process the text documents in their original form, as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length. Therefore, during the preprocessing step, the texts are converted to a more manageable representation.

One common approach for extracting features from text is to use **the bag of words model**: a model where for each document, a complaint narrative in our case, the presence (and often the frequency) of words is taken into consideration, but the order in which they occur is ignored.

The bag of words model is implemented via **CountVectorizer** where it creates a feature vector for each row.

Using CountVectorizer we successfully create our dataset and labels (**X** & **Y** lists) in order to be ready for next step.

Dataset Balancing [Imbalance Learning]

Because the dataset is not equally class distributed, we need some data balancing.

Various data balancing techniques exists:

Resampling data set (Over & Under sampling) – Penalized Models – Weight Balancing ..etc..

I've chosen Resampling the data set Techniques…

Selecting the proper class weights can sometimes be complicated. Doing a simple inverse-frequency might not always work very well. Focal loss can help, but even that will down-weight all well-classified examples of *each class equally*. Thus, another way to balance our data is by doing so *directly*, via sampling.

Under sampling means we will select only *some* of the data from the majority class, only using as many examples as the minority class has. This selection should be done to maintain the probability distribution of the class. That was easy! We just evened out our dataset by just taking less samples!

Over sampling means that we will *create copies* of our minority class in order to have the same number of examples as the majority class has. The copies will be made such that the distribution of the minority class is maintained. We just evened out our dataset without getting any more data! Sampling can be a good alternative to class balancing if you find that the class weights are difficult to set effectively.

I've chosen **Oversampling** to implement it because I'm seeing that implementing Under sampling will greatly minimize our dataset that will make any classifier working on overfitting the dataset.

Oversampling should be done after vectorizing the text data but before fitting the classifier with **SMOTE** Algorithm.

By oversampling only on the training data, none of the information in the validation data is being used to create synthetic observations. So, these results should be generalizable.

With this procedure, our approach for any classifier won't fail as we have equal enough data for all classes and our predictions won't be biased to a specific class.

What are the limitations of your methodology or Where does your approach fail? (e.g. your predictions are biased because you do not have enough data for a certain class)

After searching and testing many Classification Algorithms, one of the Most successful algorithms to solve this problem:

Multinomial Naïve Bayes Classifier

Multinomial Naïve Bayes has many advantages:

1. Give reasonable Accuracy.
2. Very fast in fitting on training data set (few seconds).
3. One of the algorithms that are easy to interpret the text.

Linear SVM Classifier

Linear SVM has many advantages:

1. Despite it takes more time than Naïve Bayes, but it provides better Accuracy Results.
2. It has many parameters to be tuned, so by choosing for them the best values yield to better results.
3. Also, the algorithm is very easy and successful to deal with vectorized text data.

Can we do better?

We can increase better our Model by **Hyperparameter Tuning.** Searching for our model's best params can achieve better results.
Here only SVM parameters can be tuned using for example **random_grid** & **RandomizedSearchCV** which get the best parameters for the classifier and do **K-Fold Cross Validation**.

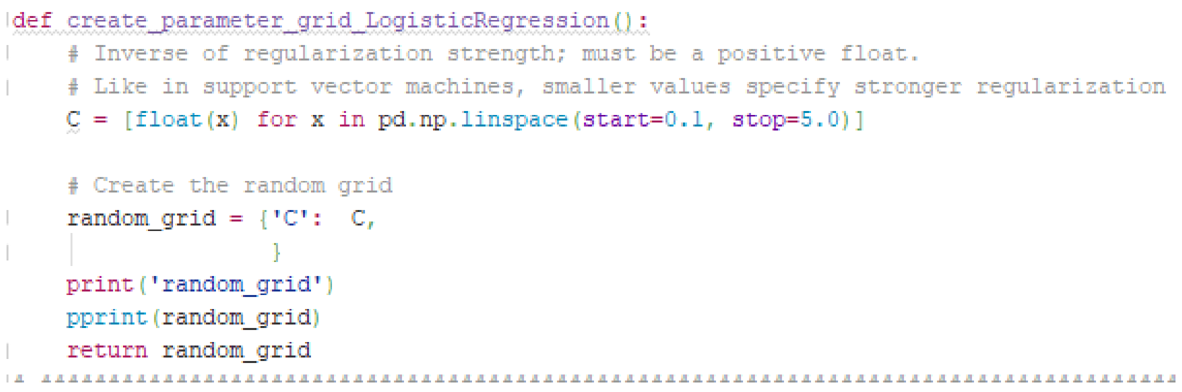
How can we use cross validation for tuning classifiers params ?

1. Put ranges for the values of params need to be tunned.
2. Create random grid for this params.
3. Use RandomizedSearchCV(estimator=rf, param_distributions=randomGrid, n_iter=30, cv=3, verbose=0, random_state=42, n_jobs=-1, refit=True)

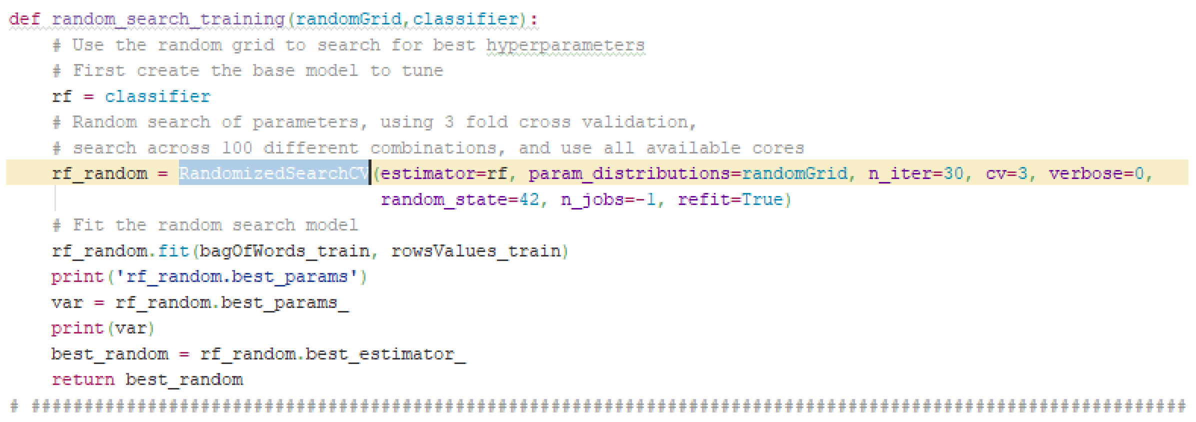
   Using here for example cross validation with k-fold = 3

4. rf_random.best_estimator_ : return best params values after doing 30 iteration using cross validation.

Here are example of Hyperparameter tuning done by me in my previous NLP Project…



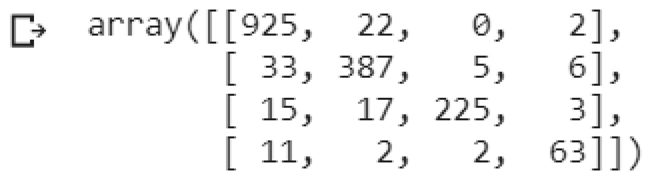
```
def create_parameter_grid_LogisticRegression():
    # Inverse of regularization strength; must be a positive float.
    # Like in support vector machines, smaller values specify stronger regularization
    C = [float(x) for x in pd.np.linspace(start=0.1, stop=5.0)]

    # Create the random grid
    random_grid = {'C':  C,
                   }
    print('random_grid')
    pprint(random_grid)
    return random_grid
# ###########################################################################
```



```
def random_search_training(randomGrid,classifier):
    # Use the random grid to search for best hyperparameters
    # First create the base model to tune
    rf = classifier
    # Random search of parameters, using 3 fold cross validation,
    # search across 100 different combinations, and use all available cores
    rf_random = RandomizedSearchCV(estimator=rf, param_distributions=randomGrid, n_iter=30, cv=3, verbose=0,
                                   random_state=42, n_jobs=-1, refit=True)
    # Fit the random search model
    rf_random.fit(bagOfWords_train, rowsValues_train)
    print('rf_random.best_params')
    var = rf_random.best_params_
    print(var)
    best_random = rf_random.best_estimator_
    return best_random
# ###########################################################################################
```

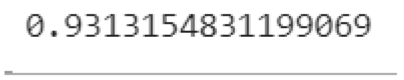
MODEL EVALUATION

Confusion Matrix



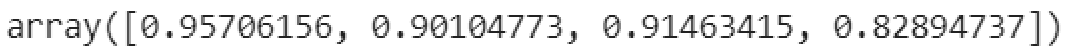
```
array([[925,  22,   0,   2],
       [ 33, 387,   5,   6],
       [ 15,  17, 225,   3],
       [ 11,   2,   2,  63]])
```

Accuracy Measure



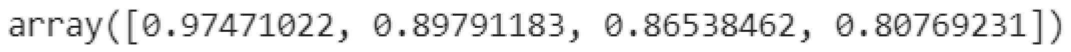
```
0.9313154831199069
```

F1 measure



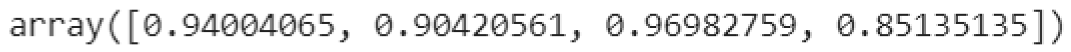
```
array([0.95706156, 0.90104773, 0.91463415, 0.82894737])
```

Recall Measure



```
array([0.97471022, 0.89791183, 0.86538462, 0.80769231])
```

Precision Measure

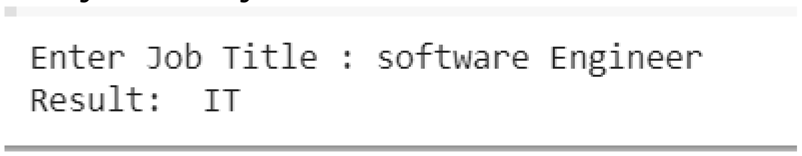


```
array([0.94004065, 0.90420561, 0.96982759, 0.85135135])
```

TESTING

New Input Test

I made a function that for a given input Job title text, outputs the predicted Industry field by the classifier



```
Enter Job Title : software Engineer
Result:  IT
```

THANK YOU