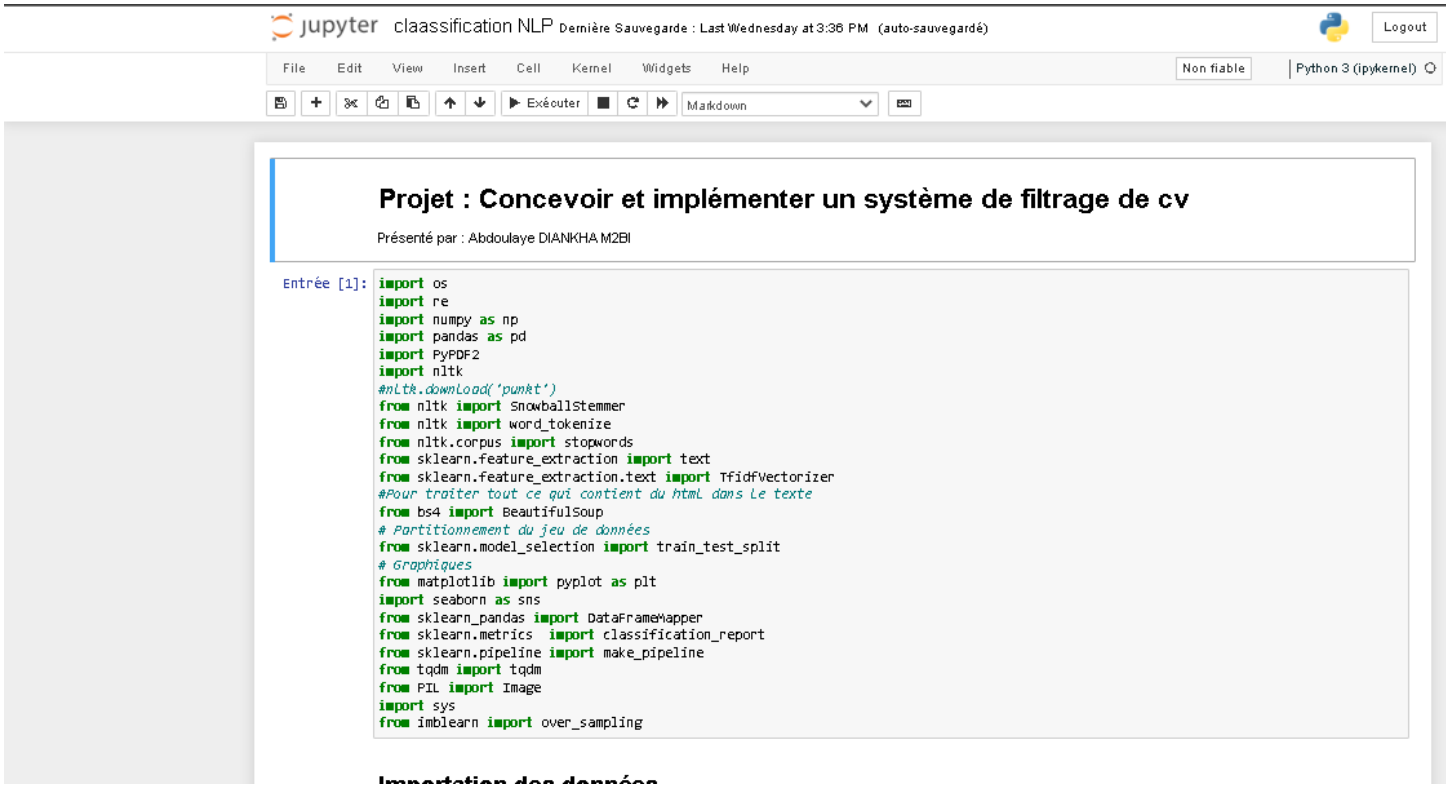
Master 2 Business Intelligence

Rapport du Projet de Texte-Mining (NLP)

Présenté et soutenu par :
Abdoulaye DIANKHA

1 : Importation des Modules nécessaires



2 : Création d'une mini fonction pour parcourir les cvs selon leur catégorie

Entrée [85]:
```
def lit_cvs(chemin=""):
    """Retourne une liste de tuples ("cv au format texte", "étiquette/classe/catégorie") contenant le jeu de données"""
    _nb = 0 #On dénombre les fichiers lus
    _categories = [_cat for _cat in os.listdir(chemin) if not(_cat.startswith("desktop"))]
    print("#Total catégories: ", len(_categories))
    _cvs = list()
    print("#Convertion des cvs de chaque catégorie...")
    for _categorie in _categories:
        _pdf_cvs= os.listdir(chemin+"/"+_categorie)
        print("Cat {} : {} => {} cvs".format(_nb, _categorie, len(_pdf_cvs)), end=" ")
        #On convertit chaque pdf
        _nb_error = 0
        for _pdf_cv in _pdf_cvs:
            try:
                pdf = PyPDF2.PdfFileReader(open(chemin+"/"+_categorie+'/'+_pdf_cv, "rb"))
            except:
                #print("error")
                _nb_error+=1
                pass
            else:
                contenu=""
                for page in pdf.pages:
                    contenu+=page.extractText()
                    #print(page.extractText())
                _cvs.append((contenu, _categorie,_pdf_cv))
        print(", {} error(s)".format(_nb_error))
        _nb+=1
    print("#Fin convertion")

    return _cvs
```

Exécution

Entrée [86]:
```
cvs = lit_cvs("./jeu_de_donnees/cv_dataset")
```

```
#Total catégories:  24
#Convertion des cvs de chaque catégorie...
Cat 0 : ACCOUNTANT => 119 cvs , 1 error(s)
Cat 1 : ADVOCATE => 119 cvs , 1 error(s)
Cat 2 : AGRICULTURE => 64 cvs , 1 error(s)
Cat 3 : APPAREL => 98 cvs , 1 error(s)
Cat 4 : ARTS => 104 cvs , 1 error(s)
```

3 : Affichage du data_frame des cvs selon leurs noms et leurs catégories

Exécution

Entrée [86]:
```
cvs = lit_cvs("./jeu_de_donnees/cv_dataset")
```

```
#Total catégories:  24
#Convertion des cvs de chaque catégorie...
Cat 0 : ACCOUNTANT => 119 cvs , 1 error(s)
Cat 1 : ADVOCATE => 119 cvs , 1 error(s)
Cat 2 : AGRICULTURE => 64 cvs , 1 error(s)
Cat 3 : APPAREL => 98 cvs , 1 error(s)
Cat 4 : ARTS => 104 cvs , 1 error(s)
Cat 5 : AUTOMOBILE => 37 cvs , 1 error(s)
Cat 6 : AVIATION => 118 cvs , 1 error(s)
Cat 7 : BANKING => 116 cvs , 1 error(s)
Cat 8 : BPO => 23 cvs , 1 error(s)
Cat 9 : BUSINESS-DEVELOPMENT => 121 cvs , 1 error(s)
Cat 10 : CHEF => 119 cvs , 1 error(s)
Cat 11 : CONSTRUCTION => 113 cvs , 1 error(s)
Cat 12 : CONSULTANT => 116 cvs , 1 error(s)
Cat 13 : DESIGNER => 108 cvs , 1 error(s)
Cat 14 : DIGITAL-MEDIA => 97 cvs , 1 error(s)
Cat 15 : ENGINEERING => 119 cvs , 1 error(s)
Cat 16 : FINANCE => 119 cvs , 1 error(s)
Cat 17 : FITNESS => 118 cvs , 1 error(s)
Cat 18 : HEALTHCARE => 116 cvs , 1 error(s)
Cat 19 : HR => 111 cvs , 1 error(s)
Cat 20 : INFORMATION-TECHNOLOGY => 121 cvs , 1 error(s)
Cat 21 : PUBLIC-RELATIONS => 112 cvs , 1 error(s)
Cat 22 : SALES => 117 cvs , 1 error(s)
Cat 23 : TEACHER => 103 cvs , 1 error(s)
#Fin convertion
```

Construction du DataFrame

Entrée [87]:
```
data=pd.DataFrame(data=cvs, columns=["cv", "categorie", "NAME"])
data
```

Out[87]:

	cv	categorie	NAME
0	ACCOUNTANTSummary\nFinancial Accountant specia...	ACCOUNTANT	10554236.pdf
1	STAFF ACCOUNTANTSummary\nHighly analytical and...	ACCOUNTANT	10674770.pdf
2	ACCOUNTANTProfessional Summary\nTo obtain a po...	ACCOUNTANT	11163645.pdf

4 : Phase de création de fonction et variable pour le prétraitement

Entrée [7]:
```
def clean (text, to_remove, regex_to_remove):
    """
        text: un document = une chaine de caractère
        to_remove: liste de mots à enlever
        regex_to_remove: une regex expression compilée

        retourne le texte pré-traité
    """
    # On décode tout ce qui est HTML
    text = BeautifulSoup(text, "lxml").text
    # On uniformise la casse de notre document en minuscule
    text = text.lower()
    #On supprimer tout ce qui est mauvais caractère
    text = regex_to_remove.sub(' ', text)
    #On supprime la liste de mots to_remove qui peut être la liste des stopwords
    text = ' '.join(word for word in text.split() if word not in to_remove)
    # On remplace chaque mots par son lemme
    WNlemming = nltk.WordNetLemmatizer()
    text = " ".join([WNlemming.lemmatize(mot, pos="v") for mot in nltk.word_tokenize(text)])
    # On peut aussi remplacer les mots plutôt par leur racine
    porter = nltk.PorterStemmer()
    text = " ".join([porter.stem(mot) for mot in nltk.word_tokenize(text)])
    return text
```

Les mots à enlever

Entrée [8]:
```
#Liste des stopwords de la langue anglaise
english_stops = list(set(stopwords.words("english")))
#On récupère les stopwords liés au jargon des cvs
with open("./jeu_de_donnees/specific_stopwords.txt") as fichier:
    specific_stops = [stop.strip() for stop in fichier.readlines()]
#Liste des stopwords
stop_words = english_stops+specific_stops
```

Les mauvais caractères

Entrée [9]:
```
Mauvais_caract = re.compile('[^0-9a-z \'#+_]')
```

Entrée [10]:
```
chemin = "./jeu_de_donnees/cv_dataset"
_categories = [_cat for _cat in os.listdir(chemin) if not(_cat.startswith("desktop"))]
```

5 : Nombre de terme obtenue avant et après nettoyage

Les mauvais caractères

Entrée [9]:
```
Mauvais_caract = re.compile('[^0-9a-z \'#+_]')
```

Entrée [10]:
```
chemin = "./jeu_de_donnees/cv_dataset"
_categories = [_cat for _cat in os.listdir(chemin) if not(_cat.startswith("desktop"))]
```

Nettoyage

Entrée [11]:
```
#Taille avant nettoyage
print("Nombre de termes avant nettoyage:  {} termes".format(len(" ".join(data.cv))))
# Nettoyage
data.cv = data.cv.apply(lambda document: clean(document, stop_words, Mauvais_caract))
#Taille après nettoyage
print("----------------------------------")
print("Nombre de termes après nettoyage:  {} termes".format(len(" ".join(data.cv))))
```

```
Nombre de termes avant nettoyage:  14957022 termes
----------------------------------
Nombre de termes après nettoyage:  8765790 termes
```

6 : Découpage de notre data_set, création Bag of words et utilisation de TF-IDF

Découpage 15% dans le testset et 75% dans le trainset et random_state=0 (on ne contole pas l'aléatoire)

Entrée [12]:
```
X_train, X_test, y_train, y_test = train_test_split(data['cv'], data['categorie'], test_size=0.15, random_state=0)
```

##Transformer

Creation du bag of words (X_vec)

Convertion des cv prétraités en une matrice cv-termes en utilisant l'approche de pondération TF-IDF (tv_matrix_trans)

Entrée [13]:
```
####Tranformer
from sklearn.feature_extraction.text import *
X_vec= TfidfVectorizer(min_df=0., max_df=1., use_idf=True).fit(X_train)
tv_matrix_trans = X_vec.fit_transform(X_train)
```

Entrée [14]:
```
####Tranformer
pipe = make_pipeline(CountVectorizer(binary=True), TfidfTransformer())
tv_matrix_trans=pipe.fit_transform(X_train)
```

Esssayons de voir si nos données d'entrainement sont équilibré

Entrée [15]:
```
y_train.value_counts()
```

Out[15]:
```
INFORMATION-TECHNOLOGY    104
FINANCE                   104
ADVOCATE                  103
BUSINESS-DEVELOPMENT      102
ACCOUNTANT                101
CONSULTANT                 99
ENGINEERING                99
AVIATION                   98
CONSTRUCTION               98
HEALTHCARE                 98
DESIGNER                   97
```

6 : Equilibrage de nos données d'entrainement

Entrée [16]:
```
tv_matrix_trans
```

Out[16]:
```
<2111x28321 sparse matrix of type '<class 'numpy.float64'>'
        with 543611 stored elements in Compressed Sparse Row format>
```

Aprés avoir fait nos transformer on élabore nos données de transformation pour équlibrer notre jeux de données

Entrée [17]:
```
from imblearn.over_sampling import RandomOverSampler
res = RandomOverSampler(sampling_strategy="not majority")
Xr,Yr = res.fit_resample(tv_matrix_trans,y_train)
```

Essayons de voir si l'"Equilibrage des données s'est éffectuez

Entrée [18]:
```
Yr.value_counts()
```

Out[18]:
```
AVIATION                104
ENGINEERING             104
CONSULTANT              104
BPO                     104
INFORMATION-TECHNOLOGY  104
SALES                   104
CHEF                    104
BUSINESS-DEVELOPMENT    104
BANKING                 104
AGRICULTURE             104
FINANCE                 104
FITNESS                 104
HEALTHCARE              104
DESIGNER                104
CONSTRUCTION            104
HR                      104
ACCOUNTANT              104
ADVOCATE                104
DIGITAL-MEDIA           104
PUBLIC-RELATIONS        104
ARTS                    104
APPAREL                 104
TEACHER                 104
AUTOMOBILE              104
Name: categorie, dtype: int64
```

6 : Affectons cet équilibrage a nos données d'entrainement

```
DIGITAL-MEDIA           104
PUBLIC-RELATIONS        104
ARTS                    104
APPAREL                 104
TEACHER                 104
AUTOMOBILE              104
Name: categorie, dtype: int64
```

Entrée [19]:
```
Xr
```

Out[19]:
```
<2496x28321 sparse matrix of type '<class 'numpy.float64'>'
        with 646674 stored elements in Compressed Sparse Row format>
```

Affectons cet équilibrage a nos données d'entrainement

Entrée [20]:
```
X_train=Xr
y_train=Yr
```

6 : Choix du bon modèle (Nous avons tester KNN et LogisticRegression)

6.1 Modèle KNN

Aprés avoir fait nos transformer et élaborer l'équlibrage de notre jeux de données

passer a l'etape de prédiction pour faire une estimation suivant le bon modéle choisit

Utilisation du Modele KNN

Entrée [21]:
```
from sklearn import neighbors
from sklearn import metrics
knn = neighbors.KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)
#testing the model
y_pred = knn.predict(X_vec.transform( X_test))
print(metrics.accuracy_score(y_test,y_pred))
```

```
0.3967828418230563
```

Entrée [22]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(neighbors.KNeighborsClassifier(3),X_train,y_train, cv=6,scoring='accuracy').mean()
```

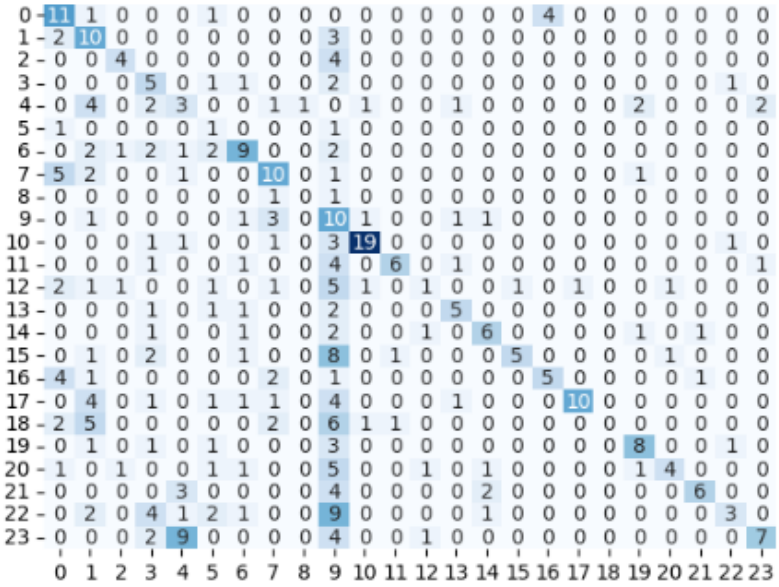
Out[22]:
```
0.4591346153846154
```

Entrée [23]:
```
classes = np.unique(y_test)
y_test_array = pd.get_dummies( y_test, drop_first=False).values

## Accuracy, Precision, Recall
accuracy = metrics.accuracy_score( y_test, y_pred)
#auc = metrics.roc_auc_score(y_test, predicted_prob, multi_class="ovr")
print("Accuracy:",  round(accuracy,2))
#print("Auc:", round(auc,2))
print("Detail:")
print(metrics.classification_report(y_test, y_pred))

## Plot confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots()
sns.heatmap(cm, annot=True, fmt='d', ax=ax, cmap=plt.cm.Blues,
            cbar=False)
```

Entrée [23]:
```
classes = np.unique(y_test)
y_test_array = pd.get_dummies( y_test, drop_first=False).values

## Accuracy, Precision, Recall
accuracy = metrics.accuracy_score( y_test, y_pred)
#auc = metrics.roc_auc_score(y_test, predicted_prob, multi_class="ovr")
print("Accuracy:",  round(accuracy,2))
#print("Auc:", round(auc,2))
print("Detail:")
print(metrics.classification_report(y_test, y_pred))

## Plot confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots()
sns.heatmap(cm, annot=True, fmt='d', ax=ax, cmap=plt.cm.Blues,
            cbar=False)
```

```
Accuracy: 0.4
Detail:
                          precision    recall  f1-score   support

              ACCOUNTANT       0.39      0.65      0.49        17
                ADVOCATE       0.29      0.67      0.40        15
             AGRICULTURE       0.57      0.50      0.53         8
                 APPAREL       0.22      0.50      0.30        10
                    ARTS       0.16      0.18      0.17        17
              AUTOMOBILE       0.08      0.33      0.13         3
                AVIATION       0.50      0.47      0.49        19
                 BANKING       0.45      0.50      0.48        20
                     BPO       0.00      0.00      0.00         2
  BUSINESS-DEVELOPMENT         0.12      0.56      0.20        18
                    CHEF       0.83      0.73      0.78        26
            CONSTRUCTION       0.75      0.43      0.55        14
              CONSULTANT       0.25      0.06      0.10        16
                DESIGNER       0.56      0.50      0.53        10
           DIGITAL-MEDIA       0.55      0.46      0.50        13
             ENGINEERING       0.83      0.26      0.40        19
                 FINANCE       0.56      0.36      0.43        14
                 FITNESS       0.91      0.43      0.59        23
              HEALTHCARE       0.00      0.00      0.00        17
                      HR       0.62      0.53      0.57        15
INFORMATION-TECHNOLOGY         0.67      0.25      0.36        16
```

Out[23]: <AxesSubplot: >



6.1 Modèle LogisticRegression en passant par gridsearchcv

utilisation du gridsearchcv afin de trouver trouver le meilleur modele avec les meilleur parameres

Essayons de trouver le meilleur model avec les meilleurs huperparamétre en comparant les différentes performance de chaque combinaison grace a la technique de cross validation

Entrée [ ]:
```
##Le modelLOgisticRegression a le meilleur score donc c le bon modèle pour notre jeux de données
```

Entrée [ ]:
```
# Grid search cross validation
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
grid={'C':np.logspace(-4,4,20),
      'penalty':['none','l1','l2'],
      'solver':['newton-cg','lbfgs','liblinear']}# L1 Lasso L2 ridge
logreg=LogisticRegression()
logreg_cv=GridSearchCV(logreg,grid,refit=True,verbose=2)
logreg_cv.fit(X_train,y_train)
logreg_cv.best_params_
```

Entrée [26]:
```
from sklearn import metrics
logreg2=LogisticRegression(C=4.281332398710396,penalty='l1', solver='liblinear', random_state=0)
logreg2.fit(X_train,y_train)
print(round(logreg2.score(X_train,y_train),2))
y_pred = logreg2.predict(X_vec.transform( X_test))
accuracy=metrics.accuracy_score(y_test,y_pred)
print("Accuracy:", round(accuracy,2))
```

```
0.9
Accuracy: 0.67
```

Entrée [28]:
```
classes = np.unique(y_test)
y_test_array = pd.get_dummies( y_test, drop_first=False).values

## Accuracy, Precision, Recall
accuracy = metrics.accuracy_score( y_test, y_pred)
#auc = metrics.roc_auc_score(y_test, predicted_prob, multi_class="ovr")
print("Accuracy:",  round(accuracy,2))
#print("Auc:", round(auc,2))
print("Detail:")
```

jupyter claassification NLP (modifié)

Logout

File Edit View Insert Cell Kernel Widgets Help

Non fiable | Python 3 (ipykernel)

Exécuter | Code

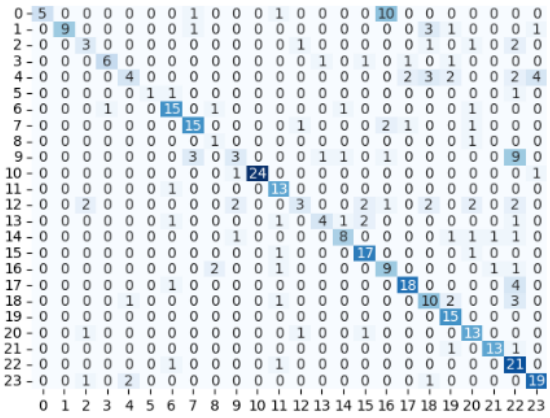
```
0.9
Accuracy: 0.67
```

Entrée [28]:

```
classes = np.unique(y_test)
y_test_array = pd.get_dummies( y_test, drop_first=False).values

## Accuracy, Precision, Recall
accuracy = metrics.accuracy_score( y_test, y_pred)
#auc = metrics.roc_auc_score(y_test, predicted_prob, multi_class="ovr")
print("Accuracy:",  round(accuracy,2))
#print("Auc:", round(auc,2))
print("Detail:")
print(metrics.classification_report(y_test, y_pred))

## Plot confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots()
sns.heatmap(cm, annot=True, fmt='d', ax=ax, cmap=plt.cm.Blues,
            cbar=False)
```

```
Accuracy: 0.67
Detail:
                         precision    recall  f1-score   support

             ACCOUNTANT       1.00      0.29      0.45        17
               ADVOCATE       1.00      0.60      0.75        15
            AGRICULTURE       0.43      0.38      0.40         8
              APPAREL         0.86      0.60      0.71        10
                ARTS          0.57      0.24      0.33        17
          AUTOMOBILE          1.00      0.33      0.50         3
            AVIATION          0.75      0.79      0.77        19
              BANKING         0.75      0.75      0.75        20
                BPO           0.25      0.50      0.33         2
  BUSINESS-DEVELOPMENT        0.43      0.17      0.24        18
                CHEF          1.00      0.92      0.96        26
          CONSTRUCTION        0.68      0.93      0.79        14
           CONSULTANT         0.50      0.19      0.27        16
             DESIGNER         0.67      0.40      0.50        10
        DIGITAL-MEDIA         0.73      0.62      0.67        13
          ENGINEERING         0.74      0.89      0.81        19
              FINANCE         0.39      0.64      0.49        14
              FITNESS         0.82      0.78      0.80        23
           HEALTHCARE         0.50      0.59      0.54        17
                 HR           0.65      1.00      0.79        15
INFORMATION-TECHNOLOGY        0.62      0.81      0.70        16
    PUBLIC-RELATIONS          0.87      0.87      0.87        15
               SALES          0.44      0.91      0.59        23
             TEACHER          0.76      0.83      0.79        23

            accuracy                               0.67       373
           macro avg         0.68      0.63      0.62       373
        weighted avg         0.70      0.67      0.65       373
```

Out[28]: <AxesSubplot: >



Résultat : Finalement nous remarquons que LogisticRegression est le meilleur modèle

Avec un score = 0.9 et accuracy =067

7 : Enregistrement du modèle et rechargement pour l'utiliser pour un new cv et prédire sa catégoriecv

Entrée [30]:
```
# Enregistrement du model
import joblib
joblib.dump(logreg2, './cv_classifier.pkl')
```

Out[30]:
```
['./cv_classifier.pkl']
```

Entrée [33]:
```
#On charge notre modèle a partir du fichier précédent
import joblib
cv_classifier_Reload = joblib.load('./cv_classifier.pkl')
```

Entrée [34]:
```
from sklearn import metrics
from sklearn.metrics import f1_score
new_fichier = "./jeu_de_donnees/cv_dataset/ACCOUNTANT/10554236.PDF"
def nouveau_cv(filename):
    new_cv = list()
    try:
        temp = open(filename, "rb")
        PDF_read = PyPDF2.PdfFileReader(temp)
    except:
          pass
    else:
        contenu=""
        first_page = PDF_read.getPage(0)
        contenu=first_page.extractText()
        new_cv.append(contenu)
    return new_cv
```

Entrée [35]:
```
new_cvs= nouveau_cv(new_fichier)
data_newcv=pd.DataFrame(data=new_cvs,columns=["cv"])
```

Affichage du nouveau CV

Entrée [37]:
```
data_newcv
```

Out[37]:

	cv
0	accountantsummari financi account special fina...

##Nettoyage


```
data_newcv=pd.DataFrame(data=new_cvs,columns=["cv"])
```

Affichage du nouveau CV

Entrée [37]:
```
data_newcv
```

Out[37]:

	cv
0	accountantsummari financi account special fina...

##Nettoyage

Entrée [40]:
```
data_newcv.cv= data_newcv.cv.apply(lambda document: clean(document, stop_words, Mauvais_caract))
```

Transformons le data_newcv pour faire un prediction

Entrée [52]:
```
CN=data_newcv.iloc[0,0]
pred = cv_classifier_Reload.predict(X_vec.transform([CN]))
print(pred[0])
```

```
ACCOUNTANT
```

NB: Pour de nouveaux exemples, penser à les suivres toutes les étapes. (conversion, prétraitement, vectorisation, etc...)

Entrée [27]:
```
""" Conclusion
decoupage d'abord de notre jeux de données
Aprés  transformation et vectorisation lorque les données sont équilibré on choisit le bon model suivant le bon score tout en utilis
xvec=countvectorize().fit(x_train)
x_train_trans=xvec.fit_transform(x_train)
choix model:
model= nommodel()
model.fit(x_train_trans,y_train)
x_test_trans=xvec.transform(x_test)
y_pred=model.predict(x_test_trans)
print(metrics.accuracy_score(y_test,y_pred))"""
```

7 : Top 10 Des meilleurs CV selon la catégorie donnée

TOp10 des CVS selon la categorie

Entrée [54]:
```
import gensim
import gensim.downloader as api
from gensim.models.doc2vec import *
import multiprocessing
cores = multiprocessing.cpu_count()
```

Entrée [89]:
```
data_cat = data[data["categorie"]=="ACCOUNTANT"]
data_cat
```

Out[89]:

	cv	categorie	NAME
0	ACCOUNTANTSummary\nFinancial Accountant specia...	ACCOUNTANT	10554236.pdf
1	STAFF ACCOUNTANTSummary\nHighly analytical and...	ACCOUNTANT	10674770.pdf
2	ACCOUNTANTProfessional Summary\nTo obtain a po...	ACCOUNTANT	11163645.pdf
3	SENIOR ACCOUNTANTExperience\nCompany Name\n \n...	ACCOUNTANT	11759079.pdf
4	SENIOR ACCOUNTANTProfessional Summary\nSenior ...	ACCOUNTANT	12065211.pdf
...	...	...	...
113	GENERAL ACCOUNTANTCareer Focus\nTo obtain a po...	ACCOUNTANT	80053367.pdf
114	SENIOR ACCOUNTANTSummary\nA highly competent, ...	ACCOUNTANT	82649935.pdf
115	PRINCIPAL ACCOUNTANTSummary\nCapable Accountan...	ACCOUNTANT	87635012.pdf
116	PAYROLL ACCOUNTANTSummary\nHas a strong work e...	ACCOUNTANT	98559931.pdf
117	ACCOUNTANT III\nSummary\nEnergetic mother of 4...	ACCOUNTANT	cv_type.pdf

118 rows × 3 columns

Entrée [90]:
```
data_cat.cv
```

Out[90]:
```
0      ACCOUNTANTSummary\nFinancial Accountant specia...
1      STAFF ACCOUNTANTSummary\nHighly analytical and...
2      ACCOUNTANTProfessional Summary\nTo obtain a po...
3      SENIOR ACCOUNTANTExperience\nCompany Name\n \n...
4      SENIOR ACCOUNTANTProfessional Summary\nSenior ...
                             ...
113    GENERAL ACCOUNTANTCareer Focus\nTo obtain a po...
114    SENIOR ACCOUNTANTSummary\nA highly competent, ...
115    PRINCIPAL ACCOUNTANTSummary\nCapable Accountan...
116    PAYROLL ACCOUNTANTSummary\nHas a strong work e...
117    ACCOUNTANT III\nSummary\nEnergetic mother of 4...
Name: cv, Length: 118, dtype: object
```

Entrée [116]:
```
def top10(categorie, n):
    data_cat = data[data["categorie"]==categorie]
    corpus = data_cat.cv
    docs=[TaggedDocument(words=word_tokenize(doc), tags=[i]) for i, doc in enumerate(corpus)]
    """Une fois formés, nous devons maintenant initialiser le modèle. cela peut être fait comme suit -"""
    model = Doc2Vec(docs, vector_size=50, window=1, min_count=1, workers=cores)
    """Maintenant, construisez le vocabulaire comme suit -"""
    model.build_vocab(docs)
    """Maintenant, entraînons le modèle Doc2Vec comme suit -"""
    model.train(docs, total_examples=model.corpus_count, epochs=100)
    """simmilarité"""
    top10 = model.dv.most_similar(len(corpus) -1, topn=n)
    return top10
```

Entrée [117]:
```
top10("ACCOUNTANT", 10)
```

Out[117]:
```
[(71, 0.7007667422294617),
 (51, 0.6904125809669495),
 (49, 0.6292073130607605),
 (67, 0.5765034556388855),
 (60, 0.5751392841339111),
 (21, 0.570796549320221),
 (107, 0.5662065744400024),
 (35, 0.5514947175979614),
 (109, 0.5499977469444275),
```

```
    model.build_vocab(docs)
    """Maintenant, entraînons le modèle Doc2Vec comme suit -"""
    model.train(docs, total_examples=model.corpus_count, epochs=100)
    """simmilarité"""
    top10 = model.dv.most_similar(len(corpus) -1, topn=n)
    return top10
```

Entrée [117]:
```
top10("ACCOUNTANT", 10)
```

Out[117]:
```
[(71, 0.7007667422294617),
 (51, 0.6904125809669495),
 (49, 0.6292073130607605),
 (67, 0.5765034556388855),
 (60, 0.5751392841339111),
 (21, 0.570796549320221),
 (107, 0.5662065744400024),
 (35, 0.5514947175979614),
 (109, 0.5499977469444275),
 (10, 0.5326189994812012)]
```

Affichage du Top10 des CVS

Entrée [125]:
```
t=[71,51,49,67,60,21,107,35,109,10]
[print(RESULTAT.iloc[doc]) for i, doc in enumerate(t)]
```

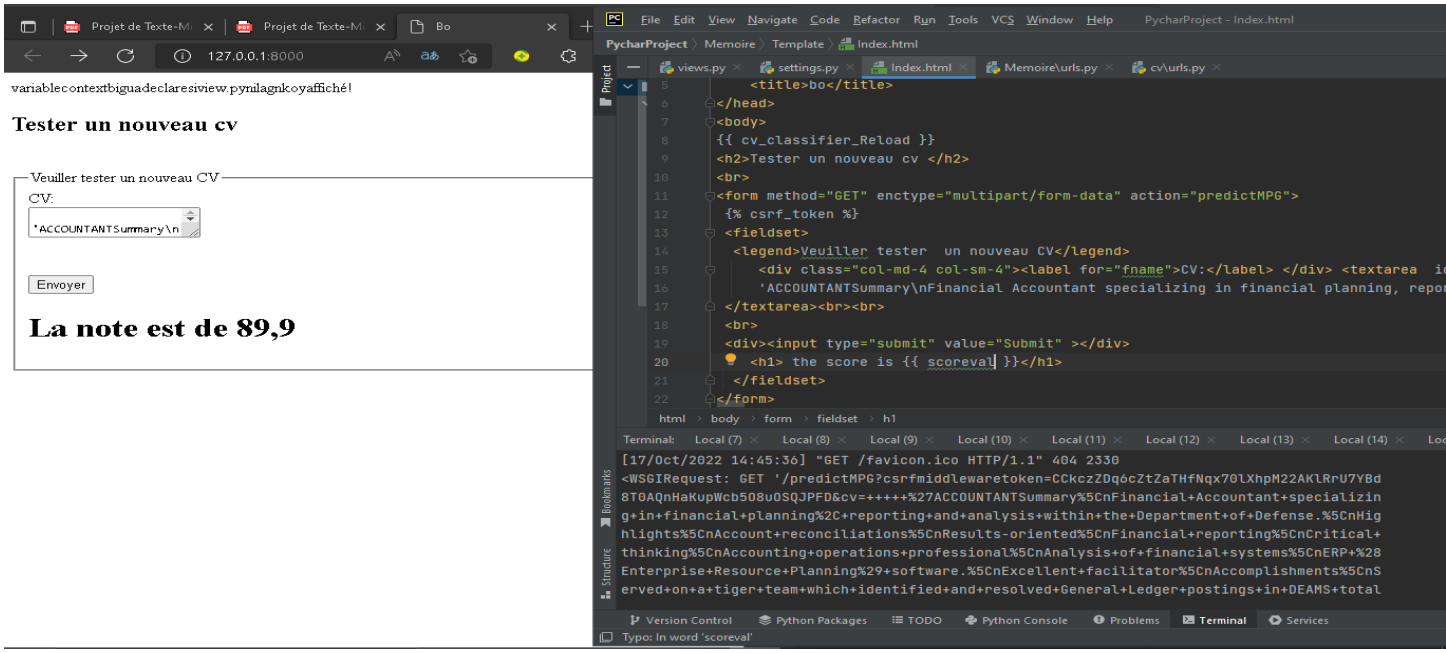
```
25935030.pdf
23246831.pdf
22925443.pdf
25749150.pdf
24703009.pdf
14496667.pdf
59403481.pdf
18669563.pdf
63137898.pdf
13072019.pdf
```

Out[125]:
```
[None, None, None, None, None, None, None, None, None, None]
```

Entrée [126]:
```
new_cvs
```

Out[126]:
```
['ACCOUNTANTSummary\nFinancial Accountant specializing in financial planning, reporting and analysis
nse.\nHighlights\nAccount reconciliations\nResults-oriented\nFinancial reporting\nCritical thinking\
```

7 : Interface Django



variablecontextbiguadeclaresiview.pynilagnkoyaffiché!

Tester un nouveau cv

Veuiller tester un nouveau CV

CV:

'ACCOUNTANTSummary\n

Envoyer

La note est de 89,9