ДОДАТОК А

Вихідний код програми

```
import re

from bs4 import BeautifulSoup
import nltk
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer

stop_words = stopwords.words('english')

def txt_replace(txt):
    if not txt:
        return ''
    try:
        html = BeautifulSoup(txt, 'html.parser')
        txt = ' '.join(html.findAll(text=True, recursive=True) or [])
        return re.sub(r"\W", " ", txt
               .replace('"', '')
               .replace('-', ' ')
               ).lower()
    except:
        return ''


def prepare_data(txt):
    words = nltk.word_tokenize(
        txt_replace(txt)
    )

    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word, pos=wordnet.VERB) for word in
words]

    no_stop_words = [word for word in words if not word in stop_words]
    return ' '.join(no_stop_words)
#%% md

Data preprocessing

#%%

import pandas as pd

from common.text_processing import prepare_data


data_df = pd.read_csv('../data/jobs.csv', skipinitialspace=True,
quotechar='"', encoding='UTF-8')

for idx, row in data_df.iterrows():
    data_df.iat[idx, 1] = prepare_data(row[1])

data_df.to_csv('../data/jobs_preprocessed.csv', index=False)
#%% md

Load Origin Data

#%%

import pandas as pd
```

```
dataset = pd.read_csv('../data/jobs.csv')
dataset.sample(15)

#%% md

Load processed data

#%%

import pandas as pd

dataset = pd.read_csv('../data/jobs_preprocessed.csv')
dataset.sample(15)


#%% md

Records counts

#%%

ds =
dataset.groupby(['title']).size().reset_index(name='counts').sort_values(by
=['counts'])
ds.head(15)

#%% md

Records count visualization

#%%

import matplotlib.pyplot as plt
prob = dataset.title.value_counts()
prob.plot(kind='bar')
plt.xticks()
plt.show()


#%% md

Word cloud counter

#%%

# grouped = dataset.groupby('title')
from wordcloud import WordCloud
wordcloud = WordCloud(width=1600, height=800, max_font_size=200,
background_color="white").generate(dataset.description[0])
plt.figure(figsize=(12,10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()

#%% md

CountVectorizer for Software Engineering (max_df = 1 min_df = 1)

#%%

from collections import defaultdict
from sklearn.feature_extraction.text import CountVectorizer
from yellowbrick.text import freqdist
```

```
# Create a dict to map target labels to documents of that category
hobbies = defaultdict(list)
for text, label in zip(dataset.description.values.astype('U'),
dataset.title):
    hobbies[label].append(text)

vectorizer = CountVectorizer(stop_words='english')
docs       = vectorizer.fit_transform(text for text in hobbies['Software
Engineering'])
features   = vectorizer.get_feature_names()
try:
    freqdist(features, docs, orient='v')
except:
    pass

#%% md

CountVectorizer for Software Engineering (max_df=0.8, min_df=2)

#%%

from collections import defaultdict
from sklearn.feature_extraction.text import CountVectorizer
from yellowbrick.text import freqdist

# Create a dict to map target labels to documents of that category
hobbies = defaultdict(list)
for text, label in zip(dataset.description.values.astype('U'),
dataset.title):
    hobbies[label].append(text)

vectorizer = CountVectorizer(max_df=0.8, min_df=2, stop_words='english')
docs       = vectorizer.fit_transform(text for text in hobbies['Software
Engineering'])
features   = vectorizer.get_feature_names()
try:
    freqdist(features, docs, orient='v')
except:
    pass

#%% md

CountVectorizer for Sales (max_df=0.8, min_df=2)

#%%

from collections import defaultdict
from sklearn.feature_extraction.text import CountVectorizer
from yellowbrick.text import freqdist

# Create a dict to map target labels to documents of that category
hobbies = defaultdict(list)
for text, label in zip(dataset.description.values.astype('U'),
dataset.title):
    hobbies[label].append(text)

vectorizer = CountVectorizer(max_df=0.8, min_df=2, stop_words='english')
docs       = vectorizer.fit_transform(text for text in hobbies['Sales'])
features   = vectorizer.get_feature_names()
try:
    freqdist(features, docs, orient='v')
except:
    pass
```

```
#%% md

TF-IDF for Sales

#%%

from collections import defaultdict
from yellowbrick.text import freqdist


# Create a dict to map target labels to documents of that category
hobbies = defaultdict(list)
for text, label in zip(dataset.description.values.astype('U'),
dataset.title):
    hobbies[label].append(text)
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(use_idf=True)
docs       = vectorizer.fit_transform(text for text in hobbies['Sales'])
features   = vectorizer.get_feature_names()
try:
    freqdist(features, docs, orient='v')
except:
    pass

#%% md

Loading dependencies

#%%

import pandas as pd
import numpy as np
from nltk.tokenize import word_tokenize
from nltk import pos_tag
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.preprocessing import LabelEncoder
from collections import defaultdict
from nltk.corpus import wordnet as wn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import model_selection, naive_bayes, svm
from sklearn.metrics import accuracy_score

#%% md

Just in case drop NA

#%%

dataset['description'].dropna(inplace=True)

#%% md

Prepare data for training

#%%

Train_X, Test_X, Train_Y, Test_Y, indices_train, indices_test =
model_selection.train_test_split(dataset['description'].values.astype('U'),
dataset['title'], dataset.index, test_size=0.3)

#%% md
```

```
Encode labels

#%%

import  joblib
Encoder = LabelEncoder()
Train_Y = Encoder.fit_transform(Train_Y)
Test_Y = Encoder.fit_transform(Test_Y)

print(f"Labels are: {Encoder.classes_}")

joblib.dump(Encoder, '../data/Encoder.pkl')

#%% md

Vectorize

#%%

Tfidf_vect = TfidfVectorizer(max_features=5000)
Tfidf_vect.fit(dataset['description'].values.astype('U'))
Train_X_Tfidf = Tfidf_vect.transform(Train_X)
Test_X_Tfidf = Tfidf_vect.transform(Test_X)

#%% md

Saving Vectors

#%%


import joblib
joblib.dump(Tfidf_vect, '../data/Tfidf_vect.pkl')
joblib.dump(Train_X_Tfidf, '../data/Train_X_Tfidf.pkl')
joblib.dump(Test_X_Tfidf, '../data/Test_X_Tfidf.pkl')

#%% md

Loading Vectors

#%%


import joblib
Tfidf_vect = joblib.load('../data/Tfidf_vect.pkl')
Train_X_Tfidf = joblib.load('../data/Train_X_Tfidf.pkl')
Test_X_Tfidf =joblib.load('../data/Test_X_Tfidf.pkl')
Encoder =joblib.load('../data/Encoder.pkl')

#%% md

Load model

#%%

import joblib
SVM = joblib.load('../data/SVM.pkl')
predictions_SVM = joblib.load('../data/predictions_SVM.pkl')

#%% md

NB
```

```
#%%

from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
# fit the training dataset on the NB classifier
Naive = naive_bayes.MultinomialNB()
Naive.fit(Train_X_Tfidf,Train_Y)
# predict the labels on validation dataset
predictions_NB = Naive.predict(Test_X_Tfidf)
# Use accuracy_score function to get the accuracy
print("Naive Bayes Accuracy Score -> ",accuracy_score(predictions_NB,
Test_Y)*100)
print("Naive Bayes Accuracy Score -> ", accuracy_score(predictions_NB,
Test_Y)*100)
print("Naive Bayes Precision Score -> ", precision_score(predictions_NB,
Test_Y, average='macro')*100 )
print("Naive Bayes FScore Score -> ", f1_score(predictions_NB, Test_Y,
average='macro')*100)
print("Naive Bayes Recall Score -> ", recall_score(predictions_NB, Test_Y,
average='macro')*100)


#%% md

SVM linear kernel

#%%

from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
# Classifier - Algorithm - SVM
# fit the training dataset on the classifier
SVM = svm.SVC(C=1.0, kernel='linear', degree=3, gamma='auto')
SVM.fit(Train_X_Tfidf,Train_Y)
# predict the labels on validation dataset
predictions_SVM = SVM.predict(Test_X_Tfidf)
# Use accuracy_score function to get the accuracy
print("SVM Accuracy Score -> ", accuracy_score(predictions_SVM,
Test_Y)*100)
print("SVM Precision Score -> ", precision_score(predictions_SVM, Test_Y,
average='macro')*100 )
print("SVM FScore Score -> ", f1_score(predictions_SVM, Test_Y,
average='macro')*100)
print("SVM Recall Score -> ", recall_score(predictions_SVM, Test_Y,
average='macro')*100)

import joblib
joblib.dump(SVM, '../data/SVM.pkl')
joblib.dump(predictions_SVM, '../data/predictions_SVM.pkl')

#%% md

Test Predictions

#%%

from common.text_processing import prepare_data
ask_result = Tfidf_vect.transform([prepare_data('''
QA experience
''')])
prediction = SVM.predict(ask_result)
print("Prediction->", Encoder.inverse_transform(prediction))

#%% md
```

```
COnfusion Matrix

#%%

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

conf_mat = confusion_matrix(Test_Y, predictions_SVM)
fig, ax = plt.subplots(figsize=(10,10))
# sns.heatmap(conf_mat, annot=True, fmt='d')
sns.heatmap(conf_mat, annot=True, fmt='d',
            xticklabels=Encoder.classes_, yticklabels=Encoder.classes_)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

#%% md

Review mistakes

#%%

np.where(Encoder.classes_=='Sales')[0][0]

#%%

from IPython.display import display
dataset['category_id'] = dataset['title'].factorize()[0]
category_id_df = dataset[['title',
'category_id']].drop_duplicates().sort_values('category_id')
category_to_id = dict(category_id_df.values)
id_to_category = dict(category_id_df[['category_id', 'title']].values)

for predicted in category_id_df.category_id:
    for actual in category_id_df.category_id:
        if predicted != actual and conf_mat[actual, predicted] >= 10:
          print("'{}' predicted as '{}' : {}
examples.".format(id_to_category[actual], id_to_category[predicted],
conf_mat[actual, predicted]))
          display(dataset.loc[indices_test[(Test_Y ==
(np.where(Encoder.classes_==id_to_category[actual])[0][0])) &
(predictions_SVM ==
(np.where(Encoder.classes_==id_to_category[predicted])[0][0]))]][['title',
'description']])
          print('')

#%% md

Feature importance

#%%

def f_importances(coef, names):
    imp = coef
    imp,names = zip(*sorted(zip(imp,names)))
    pyplot.barh(range(len(names)), imp, align='center')
    pyplot.yticks(range(len(names)), names)
    pyplot.show()
# f_importances(SVM.coef_, [1, 2 ,3 ,4 ,5 ,6, 7, 8, 9 ,0, 11, 12 ,13 ,14
,15])

#%% md
```

```
SVM rbf kernel

#%%

from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
# Classifier - Algorithm - SVM
# fit the training dataset on the classifier
SVM = svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto')
SVM.fit(Train_X_Tfidf,Train_Y)
# predict the labels on validation dataset
predictions_SVM = SVM.predict(Test_X_Tfidf)
# Use accuracy_score function to get the accuracy
print("SVM Accuracy Score -> ", accuracy_score(predictions_SVM,
Test_Y)*100)
print("SVM Precision Score -> ", precision_score(predictions_SVM, Test_Y,
average='macro')*100 )
print("SVM FScore Score -> ", f1_score(predictions_SVM, Test_Y,
average='macro')*100)
print("SVM Recall Score -> ", recall_score(predictions_SVM, Test_Y,
average='macro')*100)

import joblib
joblib.dump(SVM, '../data/SVM_rbf.pkl')
joblib.dump(predictions_SVM, '../data/predictions_SVM_rbf.pkl')

#%% md

SVM poly kernel

#%%

from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
# Classifier - Algorithm - SVM
# fit the training dataset on the classifier
SVM = svm.SVC(C=1.0, kernel='poly', degree=3, gamma='auto')
SVM.fit(Train_X_Tfidf,Train_Y)
# predict the labels on validation dataset
predictions_SVM = SVM.predict(Test_X_Tfidf)
# Use accuracy_score function to get the accuracy
print("SVM Accuracy Score -> ", accuracy_score(predictions_SVM,
Test_Y)*100)
print("SVM Precision Score -> ", precision_score(predictions_SVM, Test_Y,
average='macro')*100 )
print("SVM FScore Score -> ", f1_score(predictions_SVM, Test_Y,
average='macro')*100)
print("SVM Recall Score -> ", recall_score(predictions_SVM, Test_Y,
average='macro')*100)

import joblib
joblib.dump(SVM, '../data/SVM_poly.pkl')
joblib.dump(predictions_SVM, '../data/predictions_SVM_poly.pkl')


#%%

import matplotlib
import matplotlib.pyplot as plt
import numpy as np


labels = ['SVM (s)', 'SVM (ns)', 'RF (s)', 'RF (ns)']
```

```
accuracy = [76.18, 72.6, 55.7, 53.8]
precesion = [67.28, 65.9, 53.76, 52.3]
recall = [68.25, 66.1, 49.25, 48.5]
fscore = [69.68, 68.5, 49.03, 48.7]

x = np.arange(len(labels))  # the label locations
width = 0.2  # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(x - width, accuracy, width, label='accuracy')
rects2 = ax.bar(x - width/2, precesion, width, label='precesion')
rects3 = ax.bar(x + width/2, fscore, width, label='fscore')
rects4 = ax.bar(x + width, recall, width, label='recall')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Scores')
ax.set_title('Scores by group and gender')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

ax.bar_label(rects1, padding=10)
ax.bar_label(rects2, padding=3)
ax.bar_label(rects3, padding=3)
ax.bar_label(rects4, padding=3)

fig.tight_layout()
fig.set_size_inches(18.5, 10.5)

plt.show()

#%%
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	80 с.
	Інші документи	
2.	Презентаційні матеріали	0 плакатів

					–
.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Пільгук Ю.Ю.			Система інтелектуального порівняння профілів для рекомендації робочих вакансій
Перевірив		Терзіян В.Я			
Н.контр.		Малєєва І.А.			
Затв.		Філатов В.О.			

Шифр групи	Код напр./спец.
СШІм-19-2	122
ХНУРЕ кафедра ШІ	