تمرین چهارم

اعضای گروه:

علی درخشش
محراب مرادزاده
ابوالفضل ملک احمدی

بخش اول – طبقه بندی سند

ما در این بخش ابتدا یک مدل بر پایه Regression Logistic , SVM-Linear , Bayes Nïve آموزش دادیم که این مدل بر روی بردار ویژگی tf-idf است.
در ابتدا داده های مورد نظر خوانده میشوند :

```
data = pd.read_csv('normalized_raw_data.csv')
data.info()
data.head()
```

سپس دسته بندی های موجود در داده ها را استخراج میکنیم :

```
data.cat1.value_counts()
```

```
real-estate            575
home-kitchen           286
electronic-devices     284
vehicles               283
Name: cat1, dtype: int64
```

در قسمت بعد در تابع preprocess کاراکتر های اضافی بدون کاربرد در ستون description را که شامل :
[:,...<>/!@#$%~{}();»«…“”"؟؛◊♦–*\+_\^\=] است را حذف میکنیم :

```
def preprocess(description):
    # Text preprocessing
    # Remove extra denotations like :
    normalized_description = re.sub('[:,...<>/!@#$%~{}();»«…“”"؟؛◊♦–\*\+_\^\=]', ' ', description)
    return(normalized_description)
```

داده های ستون description بعد از حذف کاراکتر های اضافی در ستون normalized_description ذخیره میشوند:

	description	cat1	normalized_description
0	hp hp probook ۶۵... با درود لپ تاپ‌های استوک مارک	electronic-devices	hp hp probook ۶۵... با درود لپ تاپ‌های استوک مارک
1	...لپ تاپ، مهندسی، گرافیکی، پر سرعت شیک و ظریف پر	electronic-devices	...لپ تاپ مهندسی گرافیکی پر سرعت شیک و ظریف پر
2	... سلام تکنو رایان واردکننده لپتاپ استوک اروپایی	electronic-devices	... سلام تکنو رایان واردکننده لپتاپ استوک اروپایی
3	...بدون واسطه به قیمت عمده خرید کنید فروشگاه لپت	electronic-devices	...بدون واسطه به قیمت عمده خرید کنید فروشگاه لپت
4	...با سلام و احترام خدمت شما کاربر گرامی. کمال ت	electronic-devices	...با سلام و احترام خدمت شما کاربر گرامی کمال ت
...	...	...	...
1423	... سلام و درود خدمت همه عزیزان نقد و اقساط حداقل	vehicles	... سلام و درود خدمت همه عزیزان نقد و اقساط حداقل
1424	...خودرو در حد صفر میباشد اتوماتیک دارای گیربکس ژ	vehicles	...خودرو در حد صفر میباشد اتوماتیک دارای گیربکس ژ
1425	...نیوفیس بسیار تمیز و بدون خط وATام وی ام ایکس۳۳	vehicles	...نیوفیس بسیار تمیز و بدون خط وATام وی ام ایکس۳۳
1426	...فوق العاده سالم و بی رنگ یک جفت باند پایونیر چ	vehicles	...فوق العاده سالم و بی رنگ یک جفت باند پایونیر چ
1427	... موتور سالم است. مشخصات آگهی را دقیق بخوانید و	vehicles	... موتور سالم است مشخصات آگهی را دقیق بخوانید و

1428 rows × 3 columns

بخش اول – طبقه بندی سند

بعد از انجام پیش پردازش های بالا بر روی داده ها به سراغ تقسیم بندی داده برای فرایند آموزش میرویم که همانطور که در شرح تمرین خواسته شده ۹۰ درصد از دادهگان به قسمت train و ۱۰ درصد از دادهگان به قسمت test اختصاص داده شد است:

```
from sklearn.model_selection import train_test_split

X = data['normalized_description']
y = data['cat1']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 1)

print("X_train shape:", X_train.shape)
print("y_train shape:", X_train.shape)

print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (1285,)
y_train shape: (1285,)
X_test shape: (143,)
y_test shape: (143,)
```

سپس از TfidfVetorizer که برای استخراج ویژگی متن استفاده میکنیم تا ویژگی های متن را استخراج کنیم :

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(lowercase = False, tokenizer = word_tokenize)

tfidf.fit(X_train)
X_train_tf_idf = tfidf.transform(X_train)
X_test_tf_idf = tfidf.transform(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/feature_extraction/text.py:528:
  warnings.warn(


Train : 90% Test : 10%
Evaluation : using 10_fold_cv
8209 features without hazm.Normalizer
8296 features with hazm.Normalizer
```

در قسمت بعد ماتریس tf-idf با استفاده از todense به نمایش dense تبدیل کرده و سپس به آرایه تبدیل میکنیم :

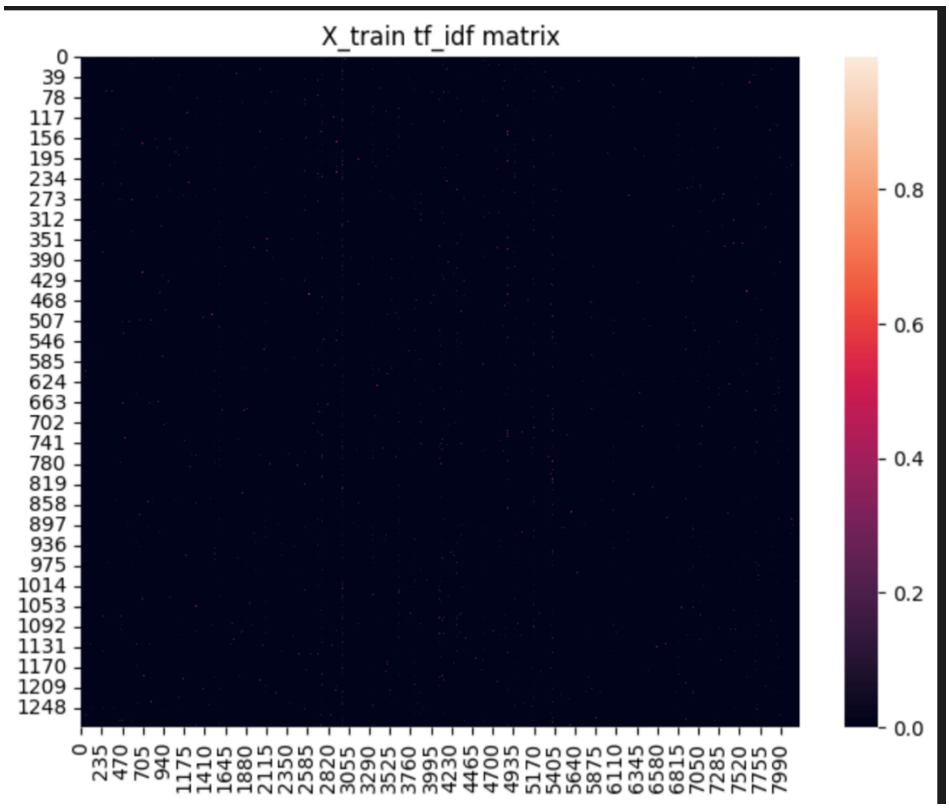
```
X_train_tf_idf = X_train_tf_idf.todense()
X_test_tf_idf = X_test_tf_idf.todense()

X_train_tf_idf = np.asarray(X_train_tf_idf)
X_test_tf_idf = np.asarray(X_test_tf_idf)


print("X_train_tf_idf shape :", X_train_tf_idf.shape)
print("X_test_tf_idf shape :", X_test_tf_idf.shape)
```

```
X_train_tf_idf shape : (1285, 8209)
X_test_tf_idf shape : (143, 8209)
```

در نهایت نمایش دادهگان ما به صورت زیر است :



** که یک ماتریس پراکنده است که ما از SVD استفاده میکنیم.

شروع کار با مدل های نام برده

در ابتدا قبل شروع فرایند یادگیری و کار کردن با مدل های گفته شد ما معیار های ارزیابی را در تابع metrics_report تعریف میکنیم :

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, ConfusionMatrixDisplay, confusion_matrix
def metrics_report(estimator, X_train, y_train, X_test, y_test, plot = True, ret = False):
  '''
  if you put ret to True it will return the following tupple:
    (accuracy_train, accuracy_test, precision_train, precision_test, recall_train, recall_test, f1_train, f1_test)
  '''
  pred_train = estimator.predict(X_train)
  pred_test = estimator.predict(X_test)

  accuracy_train = accuracy_score(y_train, pred_train)
  accuracy_test = accuracy_score(y_test, pred_test)

  precision_train = precision_score(y_train, pred_train, average = 'macro')
  precision_test = precision_score(y_test, pred_test, average = 'macro')

  recall_train = recall_score(y_train, pred_train, average = 'macro')
  recall_test = recall_score(y_test, pred_test, average = 'macro')

  f1_train = f1_score(y_train, pred_train, average = 'macro')
  f1_test = f1_score(y_test, pred_test, average = 'macro')

  if ret:
    return (accuracy_train, accuracy_test, precision_train, precision_test, recall_train, recall_test,
            f1_train, f1_test)

  else :
    print("accuracy on train = {:.3f}%".format(accuracy_train*100))
    print("accuracy on test = {:.3f}%".format(accuracy_test*100))

    print("\n")

    print("precision(macro) on train = {:.4f}".format(precision_train))
    print("precision(macro) on test = {:.4f}".format(precision_test))

    print("\n")

    print("recall(macro) on train = {:.4f}".format(recall_train))
    print("recall(macro) on test = {:.4f}".format(recall_test))

    print("\n")

    print("f1(macro) on train = {:.4f}".format(f1_train))
    print("f1(macro) on test = {:.4f}".format(f1_test))

    print("\n")

  if plot:
    disp = ConfusionMatrixDisplay(confusion_matrix(y_test, pred_test),
                                  display_labels = np.unique(y_test))
    fig, ax = plt.subplots(figsize=(8, 8))
    disp.plot(ax=ax, cmap = 'rocket')
    plt.show()
```

تعریف کردن مدل های مورد استفاده :

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import TruncatedSVD
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

scaler = StandardScaler()
svd = TruncatedSVD(random_state = 1)
naive_bayes = GaussianNB()
log_reg = LogisticRegression(random_state = 1, max_iter = 500, penalty = 'l2')
svm = SVC(random_state = 1)

pipe_naive_bayes = Pipeline(steps=[("svd", svd) , ("naive_bayes", naive_bayes)])
pipe_log_reg = Pipeline(steps=[("scaler", scaler), ("svd", svd) , ("log_reg", log_reg)])
pipe_svm = Pipeline(steps=[("scaler", scaler), ("svd", svd) , ("svm", svm)])
```

در ادامه ما برای یافتن بهترین هایپرپارامتر های ممکن از GridSearchCV استفاده میکنیم که در ابتدا در قسمت زیر مقادیر مورد نظر برای هایپرپارامتر های موجود برای هر مدل تعریف شده است :

- از pipe line استفاده شده که تعداد ابعاد svd هم با توجه به هر مدل tune شده.

```
params_naive_bayes = {
    "svd__n_components": [100, 250 ,500],
}

params_log_reg = {
    "svd__n_components": [100, 250 ,500],
    "log_reg__C": [0.1, 1, 10],
    "log_reg__class_weight": [None, 'balanced']
}

(variable) params_svm: dict[str, Any]
params_svm = {
    "svd__n_components": [100, 250 ,500],
    'svm__C': [0.1, 1, 10],
    'svm__gamma': [0.1, 1, 10],
    'svm__kernel': ['linear', 'poly'], # rbf is not good for this
    "svm__class_weight": [None, 'balanced']
}
```

پارامتر های استخراج شده برای Naive bayesبه صورت زیر است :

```
grid_search_naive_bayes = GridSearchCV(pipe_naive_bayes, params_naive_bayes, cv=5, verbose = 5)
grid_search_naive_bayes.fit(X_train_tf_idf, y_train)
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV 1/5] END .............svd__n_components=100;, score=0.918 total time=   3.4s
[CV 2/5] END .............svd__n_components=100;, score=0.899 total time=   3.1s
[CV 3/5] END .............svd__n_components=100;, score=0.856 total time=   2.8s
[CV 4/5] END .............svd__n_components=100;, score=0.930 total time=   3.2s
[CV 5/5] END .............svd__n_components=100;, score=0.938 total time=   2.2s
[CV 1/5] END .............svd__n_components=250;, score=0.626 total time=   4.0s
[CV 2/5] END .............svd__n_components=250;, score=0.732 total time=   4.3s
[CV 3/5] END .............svd__n_components=250;, score=0.634 total time=   8.1s
[CV 4/5] END .............svd__n_components=250;, score=0.545 total time=   7.9s
[CV 5/5] END .............svd__n_components=250;, score=0.860 total time=   6.1s
[CV 1/5] END .............svd__n_components=500;, score=0.381 total time=   8.2s
[CV 2/5] END .............svd__n_components=500;, score=0.428 total time=   7.7s
[CV 3/5] END .............svd__n_components=500;, score=0.385 total time=   8.8s
[CV 4/5] END .............svd__n_components=500;, score=0.370 total time=   7.6s
[CV 5/5] END .............svd__n_components=500;, score=0.374 total time=   8.7s
```

GridSearchCV

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('svd', TruncatedSVD(random_state=1)),
                                       ('naive_bayes', GaussianNB())]),
             param_grid={'svd__n_components': [100, 250, 500]}, verbose=5)
```

estimator: Pipeline

TruncatedSVD

GaussianNB

```
grid_search_naive_bayes.best_score_, grid_search_naive_bayes.best_params_
```

```
(0.9081712062256809, {'svd__n_components': 100})
```

سپس بعد از بدست آوردن هایپرپارامتر های مناسب آنها را در مدل جایگزاری میکنیم و دقت مدل را با استفاده از معیار های ارزیابی تعریف شده محاسبه میکنم:

```
tuned_naive_bayes = Pipeline(steps=[("svd", TruncatedSVD(random_state = 1, n_components = 100)) ,
                ("naive_bayes", GaussianNB())])
tuned_naive_bayes.fit(X_train_tf_idf, y_train)
```

Pipeline

TruncatedSVD

GaussianNB

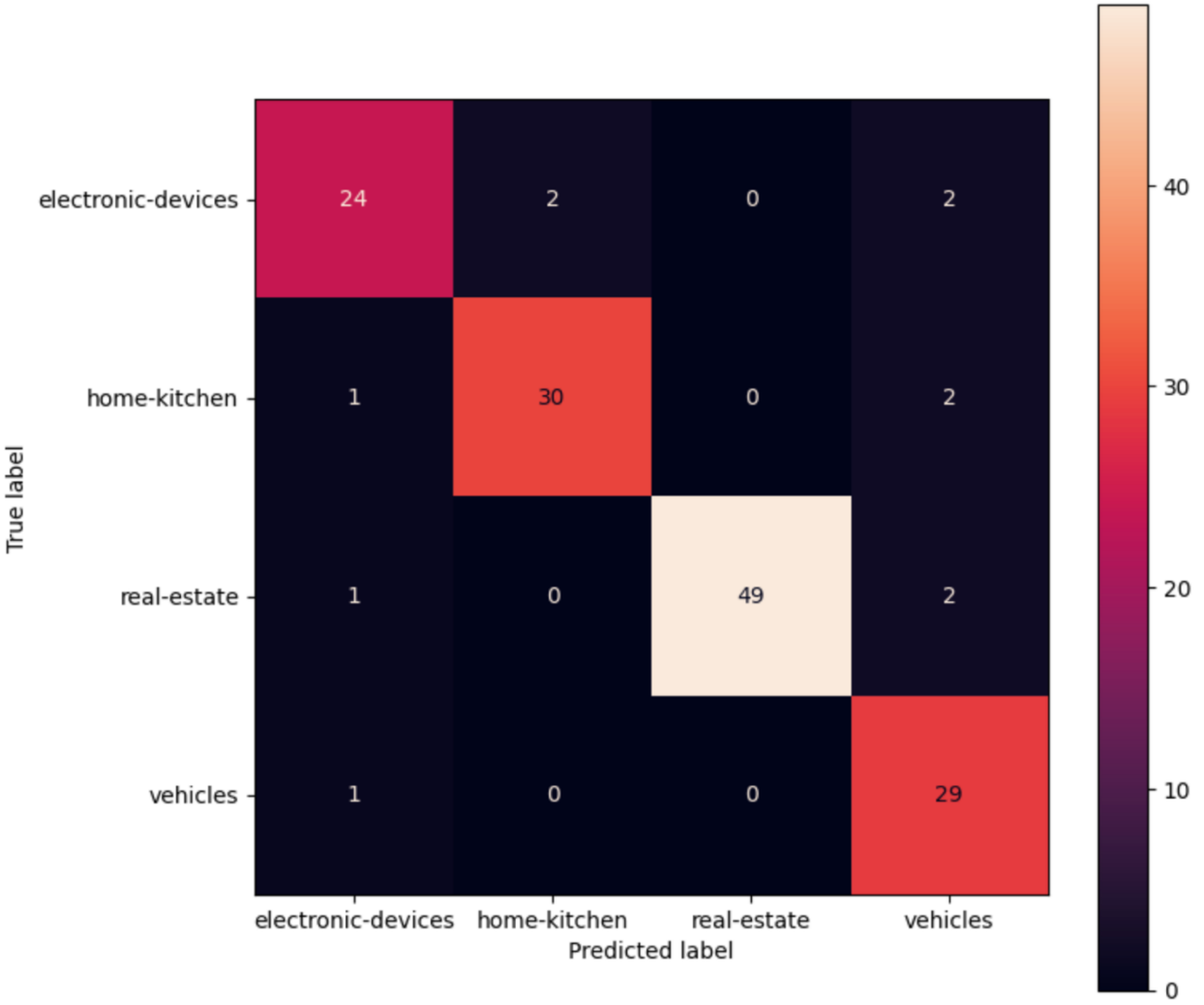
```
metrics_report(tuned_naive_bayes, X_train_tf_idf, y_train, X_test_tf_idf, y_test, plot = True, ret = False)
```

```
accuracy on train = 92.685%
accuracy on test = 92.308%

precision(macro) on train = 0.9126
precision(macro) on test = 0.9137

recall(macro) on train = 0.9182
recall(macro) on test = 0.9188

f1(macro) on train = 0.9146
f1(macro) on test = 0.9146
```



در ادامه مراحل بالا را برای مدل logistic regression و SVM نیز تکرار شده نتایج در زیر گزارش میشوند :

Logistic regression:

```
grid_search_log_reg.best_score_, grid_search_log_reg.best_params_
```

```
(0.9548638132295719,
 {'log_reg__C': 0.1, 'log_reg__class_weight': None, 'svd__n_components': 500})
```

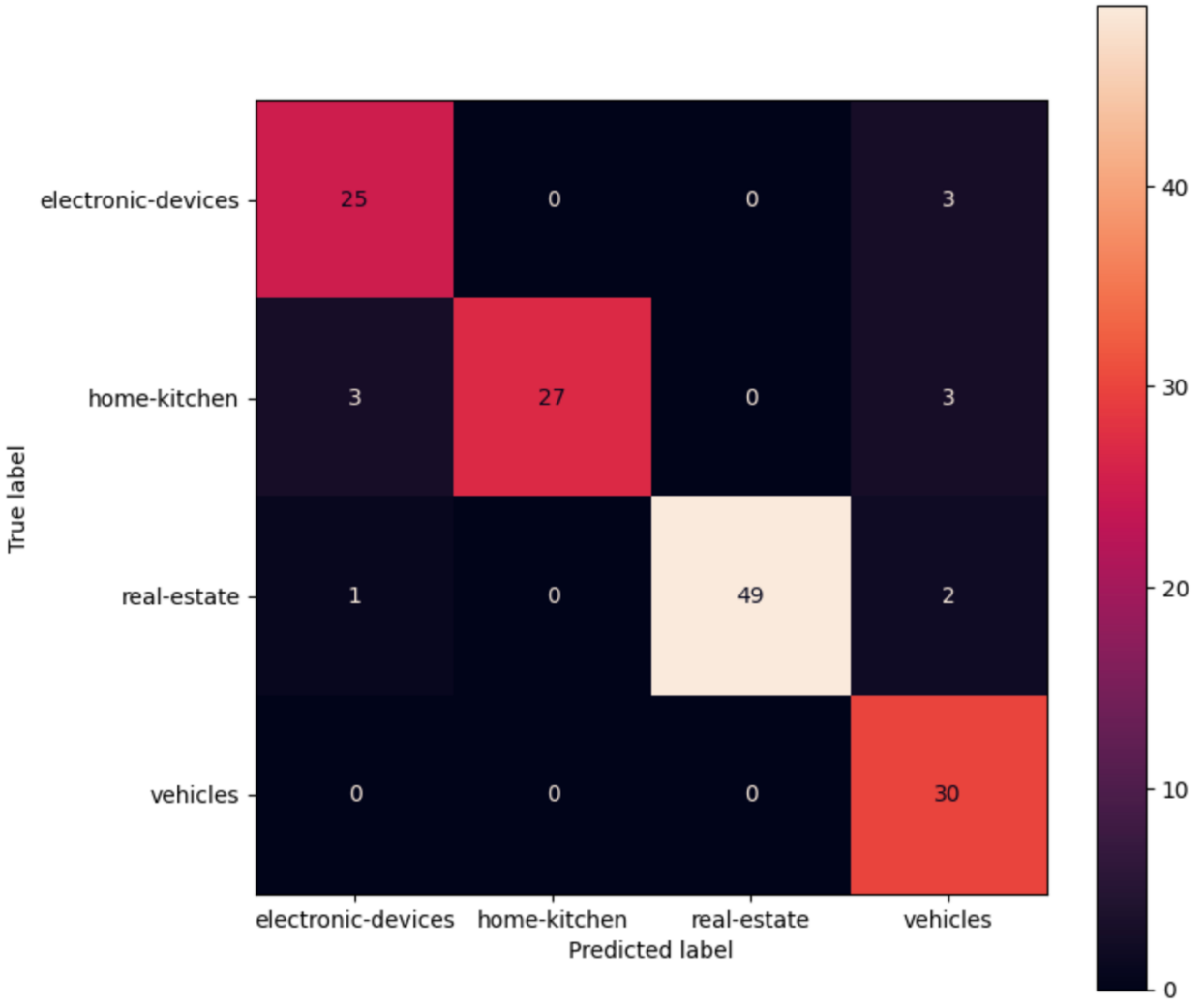
```
metrics_report(tuned_log_reg, X_train_tf_idf, y_train, X_test_tf_idf, y_test, plot = True, ret = False)
```

```
accuracy on train = 99.533%
accuracy on test = 91.608%


precision(macro) on train = 0.9946
precision(macro) on test = 0.9129


recall(macro) on train = 0.9941
recall(macro) on test = 0.9133


f1(macro) on train = 0.9943
f1(macro) on test = 0.9075
```



SVM:

```
grid_search_svm = GridSearchCV(pipe_svm, params_svm, cv=5, verbose = 5)
grid_search_svm.fit(X_train_tf_idf, y_train)
```

```
grid_search_svm.best_score_, grid_search_svm.best_params_
```

```
(0.9299610894941635,
 {'svd__n_components': 500,
  'svm__C': 0.1,
  'svm__class_weight': 'balanced',
  'svm__gamma': 0.1,
  'svm__kernel': 'linear'})
```

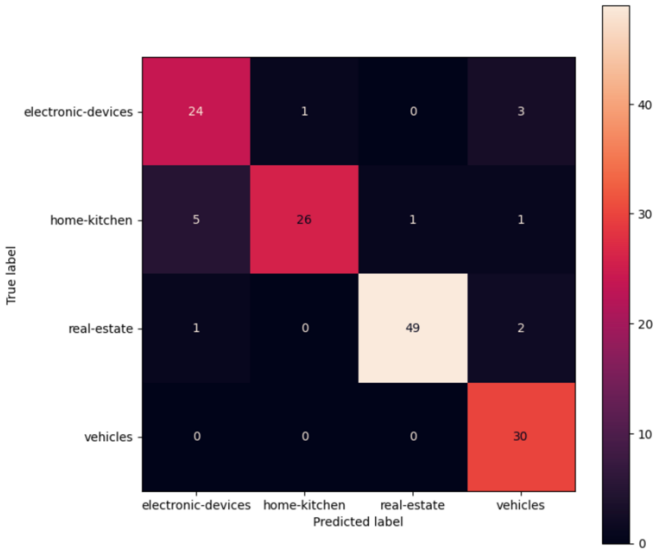
```
metrics_report(tuned_svm, X_train_tf_idf, y_train, X_test_tf_idf, y_test, plot = True, ret = False)
```

```
accuracy on train = 99.844%
accuracy on test = 90.210%


precision(macro) on train = 0.9980
precision(macro) on test = 0.8941


recall(macro) on train = 0.9980
recall(macro) on test = 0.8968


f1(macro) on train = 0.9980
f1(macro) on test = 0.8910
```



مدل ترنسفورمری

در ابتدا سعی کردیم تا دیتاست را دقیقا تبدیل به فرمت دیتاست‌های سایت huggingface.co کنیم تا به‌راحتی بتوان توسط مدل‌های این سایت استفاده کرد و در ترم‌های آینده استاد بتوانند از این دیتاست در نوت‌بوک‌های خود به راحتی استفاده کنند. همچنین این دیتاست را در سایت huggingface.co به زودی بارگزاری خواهیم کرد.

در مرحله اول علائم نگارشی را از متون حذف می‌کنیم و با استفاده از نرمالایزر کتابخانه hazm جملات را نرمال می‌کنیم.

```
dataset = pd.read_csv('normalized_raw_data.csv')
data = dataset['description'].tolist()
labels = dataset['cat1'].tolist()
```

```
from hazm import sent_tokenize, Normalizer
import re

print('orgi len', len(data))
normalized_description_list = []
normalizer = Normalizer()
for description in tqdm(data, desc = 'Normalization'):
    normalized_description = re.sub('[:,،.<>/!@#$%~{}();»«…“”"؟:۰◆-\*\+_\^]', ' ', description)
    normalized_description = normalizer.normalize(normalized_description)
    normalized_description_list.append(normalized_description)
print('norm len: ', len(normalized_description_list))
```

لیبل‌ها را به اعداد مپ می‌کنیم، همانند دیتاست‌های huggingface

```
# Define the mapping from strings to numbers
mapping = {
    "electronic-devices": 0,
    "vehicles": 1,
    "real-estate": 2,
    "home-kitchen": 3
}

# Convert strings to numbers based on the mapping
labels = [mapping[item] for item in labels]
```

دیتا و لیبل را در یک دیتافریم قرار داده و با استفاده از model_selection به ۳ قسمت test,train و validation تقسیم می‌کنیم.

```
from sklearn.model_selection import train_test_split

# Create a DataFrame
df = pd.DataFrame({'data': normalized_description_list, 'labels': labels})

# Split the DataFrame into train, test, and validation sets
train_df, test_val_df = train_test_split(df, test_size=0.2, random_state=42)
test_df, val_df = train_test_split(test_val_df, test_size=0.5, random_state=42)

# Reset the index of the DataFrames
train_df.reset_index(drop=True, inplace=True)
test_df.reset_index(drop=True, inplace=True)
val_df.reset_index(drop=True, inplace=True)

# Print the number of samples in each set
print("Train set size:", len(train_df))
print("Test set size:", len(test_df))
print("Validation set size:", len(val_df))
```

با استفاده از توابع Dataset و DatasetDict از Huggingface دیتاست را به فرمت موردنظر تبدیل می‌کنیم.

```
from datasets import Dataset, DatasetDict

import datasets
import pandas as pd

datasets_train_test = DatasetDict({
    "train": Dataset.from_pandas(train_df),
    "test": Dataset.from_pandas(test_df),
    "validation": Dataset.from_pandas(val_df)
    })
```

نتایج ساخت دیتاست:

```
DatasetDict({
    train: Dataset({
        features: ['data', 'labels'],
        num_rows: 1142
    })
    test: Dataset({
        features: ['data', 'labels'],
        num_rows: 143
    })
    validation: Dataset({
        features: ['data', 'labels'],
        num_rows: 143
    })
})
```

توکنایزر و مدل:

ما از مدل ParsBert v۳ و توکنایزر آن استفاده کرده‌ایم.
در ابتدا با استفاده از توکنایزر و کتابخانه map که سرعت پردازش را افزایش می‌دهد دیتا را truncation می‌کنیم و attention_mask را محاسبه می‌کنیم.

```
def preprocess_function(examples):
    return tokenizer(examples["data"], truncation=True)

tokenized_dataset = datasets_train_test.map(preprocess_function, batched=True)
```

```
Map:   0%|
Map:  88%|████████████████████████████████████████          | 1000/1142

Map:   0%|

Map:   0%|
```

```
tokenized_dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['data', 'labels', 'input_ids', 'token_type_ids', 'attention_mask'],
        num_rows: 1142
    })
    test: Dataset({
        features: ['data', 'labels', 'input_ids', 'token_type_ids', 'attention_mask'],
        num_rows: 143
    })
    validation: Dataset({
        features: ['data', 'labels', 'input_ids', 'token_type_ids', 'attention_mask'],
        num_rows: 143
    })
})
```

تابع compute_metrics معیارهای ارزیابی خواسته شده در این تسک را برای ما محاسبه می‌کند.

```
from datasets import load_metric
import numpy as np

metric1 = load_metric("precision")
metric2 = load_metric("recall")
metric3 = load_metric("f1")
metric4 = load_metric("accuracy")
metric5 = evaluate.load("BucketHeadP65/confusion_matrix")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)

    precision = metric1.compute(predictions=predictions, references=labels, average="micro")["precision"]
    recall = metric2.compute(predictions=predictions, references=labels, average="micro")["recall"]
    f1_micro = metric3.compute(predictions=predictions, references=labels, average="micro")["f1"]
    f1_macro = metric3.compute(predictions=predictions, references=labels, average="macro")["f1"]
    accuracy = metric4.compute(predictions=predictions, references=labels)["accuracy"]
    confusion_matrix = metric5.compute(predictions=predictions, references=labels)["confusion_matrix"]

    return {"precision": precision, "recall": recall, "f1_micro": f1_micro, "f1_macro": f1_macro, "accuracy": accuracy, "confusion_matrix": confusion_matrix}
```

با استفاده از توابع training_args و Trainer مدل خود را فاین‌تیون می‌کنیم. هایپرپارامترهای مدل در عکس زیر مشخص شده اند.

```python
training_args = TrainingArguments(
    output_dir="ParsBERT_V3_results",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    # evaluation_strategy="epoch",
    save_strategy="epoch",
    # load_best_model_at_end=True,
    logging_steps=20,
    save_steps=50,
    logging_dir='./ParsBERT_V3_logs'
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["validation"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
```

نتایج پس از ۵ ایپاک به صورت زیر می‌باشد.

```
trainer.train()
```

```
/home/user01/miniconda3/lib/python3.7/site-packages/transformers/optimization.py:395: FutureWarning: This implementation of AdamW is deprecated and will be removed in a future version. Use the PyTorch implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True` to disable this warning
  FutureWarning,
```

[216/216 00:52, Epoch 3/3]

Step	Training Loss
20	0.039600
40	0.013800
60	0.053300
80	0.011100
100	0.001500
120	0.011600
140	0.017800
160	0.001400
180	0.004500
200	0.012900

```
TrainOutput(global_step=216, training_loss=0.015668955967865057, metrics={'train_runtime': 52.7772, 'train_samples_per_second': 64.914, 'train_steps_per_second': 4.093, 'total_flos': 315873577393296.0, 'train_loss': 0.015668955967865057, 'epoch': 3.0})
```

دیتاست

در این بخش ما در ابتدا دیتاست آگهی‌های بخش خانه وبسایت دیوار را مانند بخش قبلی نرمال کرده و علائم نگارشی را حذف نمودیم. سپس با استفاده از ابزار label studio به صورت دستی حدود **۲۳ هزار کلمه** را tag زدیم. سپس دیتاستی به صورت conell۲۰۰۳ بدست آوردیم. هر پاراگراف در این دیتاست با استفاده از \n\n جدا شده است.

```
word  tag
۵  O
طبقه  O
۳  B-Attributes of the property (A)
واحدی  I-Attributes of the property (A)
طبقه  B-Attributes of the property (A)
دوم  I-Attributes of the property (A)
۲  B-Attributes of the property (A)
خوابه  I-Attributes of the property (A)
۲  B-Attributes of the property (A)
سرویس  I-Attributes of the property (A)
بهداشتی  I-Attributes of the property (A)
آشپزخانه  B-Attributes of the property (A)
اوپن  I-Attributes of the property (A)
بدونه  B-Attributes of the property (A)
دیوار  I-Attributes of the property (A)
مشترک  I-Attributes of the property (A)
پارکینگ  B-Attributes of the property (A)
```

همانند بخش قبلی سعی کردیم تا دیتاست را دقیقا تبدیل به فرمت دیتاست‌های سایت huggingface.co کنیم تا به‌راحتی بتوان توسط مدل‌های این سایت استفاده کرد و در ترم‌های آینده استاد بتوانند از این دیتاست در نوت‌بوک‌های خود به راحتی استفاده کنند. این دیتاست نیز در سایت huggingface.co به زودی بارگزاری خواهیم کرد تا در اختیار عموم قرار گیرد.

اضافه کردن کلمات هر جمله و جملات به یک لیست ۲ بعدی و همین طور تگ ها به لیست ۲ بعدی:

```python
sentences = []
sentence = []
list_tags = []
list_tag = []
for index, row in dataset.iterrows():
    if pd.isna(row['word']):
        sentences.append(sentence)
        sentence = []
        list_tags.append(list_tag)
        list_tag = []
        pass
    else:
        sentence.append(row['word'])
        list_tag.append(row['tag'])
```

مپ کردن تگ‌ها به اعداد و ساخت یک دیتافریم شامل جملات و تگ‌های NER

```
# Define the mapping from strings to numbers
mapping = {
    "O": 0,
    "B-Locality (L)": 1,
    "I-Locality (L)": 2,
    "B-Total Price (P)": 3,
    "I-Total Price (P)": 4,
    "B-Land Area (LA)": 5,
    "I-Land Area (LA)": 6,
    "B-Cost per land area (C)": 7,
    "I-Cost per land area (C)": 8,
    "B-Contact name (N)": 9,
    "I-Contact name (N)": 10,
    "B-Contact telephone (T)": 11,
    "I-Contact telephone (T)": 12,
    "B-Attributes of the property (A)": 13,
    "I-Attributes of the property (A)": 14
}

# Convert strings to numbers based on the mapping
list_tags = [[mapping[item] for item in list_tag] for list_tag in list_tags]
```

```
df = pd.DataFrame({'tokens': sentences, 'ner_tags': list_tags})
df.head()
```

داده‌ها را با استفاده از model_selection به ۳ قسمت test,train و validation تقسیم می‌کنیم. و سپس با استفاده از تابع DatasetDict دیتاست را به فرم دیتاست‌های huggingface درمی‌آوریم.

```python
from sklearn.model_selection import train_test_split

# Split the DataFrame into train, test, and validation sets
train_df, test_val_df = train_test_split(df, test_size=0.2, random_state=42)
test_df, val_df = train_test_split(test_val_df, test_size=0.5, random_state=42)

# Reset the index of the DataFrames
train_df.reset_index(drop=True, inplace=True)
test_df.reset_index(drop=True, inplace=True)
val_df.reset_index(drop=True, inplace=True)

# Print the number of samples in each set
print("Train set size:", len(train_df))
print("Test set size:", len(test_df))
print("Validation set size:", len(val_df))
```

```
Train set size: 446
Test set size: 56
Validation set size: 56
```

```python
from datasets import Dataset, DatasetDict

import datasets
import pandas as pd

datasets_train_test = DatasetDict({
    "train": Dataset.from_pandas(train_df),
    "test": Dataset.from_pandas(test_df),
    "validation": Dataset.from_pandas(val_df)
    })
```

فرمت دیتاست را در زیر مشاهده می‌کنید:

```
DatasetDict({
    train: Dataset({
        features: ['tokens', 'ner_tags'],
        num_rows: 446
    })
    test: Dataset({
        features: ['tokens', 'ner_tags'],
        num_rows: 56
    })
    validation: Dataset({
        features: ['tokens', 'ner_tags'],
        num_rows: 56
    })
})
```

جملات و لیبل‌های کلمات را با استفاده از تابع zip به هم پیوند میزنیم.

```
words = raw_datasets["train"][0]["tokens"]
labels = raw_datasets["train"][0]["ner_tags"]
line1 = ""
line2 = ""
for word, label in zip(words, labels):
    full_label = label_names[label]
    max_length = max(len(word), len(full_label))
    line1 += word + " " * (max_length - len(word) + 1)
    line2 += full_label + " " * (max_length - len(full_label) + 1)

print(line1)
print(line2)
```

توکنایزر و مدل:

ما از مدل ParsBert v۳ و توکنایزر آن استفاده کرده‌ایم.

در ابتدا لیبل‌ها و توکن‌ها را با هم align می‌کنیم. همچنین توکن‌های خاص مانند [sep] و [cls] را با عدد ۱۰۰- جایگزین می‌کنیم.

```
def align_labels_with_tokens(labels, word_ids):
    new_labels = []
    current_word = None
    for word_id in word_ids:
        if word_id != current_word:
            # Start of a new word!
            current_word = word_id
            label = -100 if word_id is None else labels[word_id]
            new_labels.append(label)
        elif word_id is None:
            # Special token
            new_labels.append(-100)
        else:
            # Same word as previous token
            label = labels[word_id]
            # If the label is B-XXX we change it to I-XXX
            if label % 2 == 1:
                label += 1
            new_labels.append(label)

    return new_labels
```

سپس همانند دیتاست قبلی با استفاده از تابع map دیتاها را به صورت بچ شده توکنایز و turncat می‌کنیم.

```
def tokenize_and_align_labels(examples):
    tokenized_inputs = tokenizer(
        examples["tokens"], truncation=True, is_split_into_words=True
    )
    all_labels = examples["ner_tags"]
    new_labels = []
    for i, labels in enumerate(all_labels):
        word_ids = tokenized_inputs.word_ids(i)
        new_labels.append(align_labels_with_tokens(labels, word_ids))

    tokenized_inputs["labels"] = new_labels
    return tokenized_inputs
```

```
tokenized_datasets = raw_datasets.map(
    tokenize_and_align_labels,
    batched=True,
    remove_columns=raw_datasets["train"].column_names,
)
```

با استفاده از تابع compute_metrics معیارهای ارزیابی را برای این تسک درست می‌کنیم. سپس با id۲label و label۲id مپینگ تگ‌ها و اعداد را درست می‌کنیم.

```
import numpy as np


def compute_metrics(eval_preds):
    logits, labels = eval_preds
    predictions = np.argmax(logits, axis=-1)

    # Remove ignored index (special tokens) and convert to labels
    true_labels = [[label_names[l] for l in label if l != -100] for label in labels]
    true_predictions = [
        [label_names[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]
    all_metrics = metric.compute(predictions=true_predictions, references=true_labels)
    return {
        "precision": all_metrics["overall_precision"],
        "recall": all_metrics["overall_recall"],
        "f1": all_metrics["overall_f1"],
        "accuracy": all_metrics["overall_accuracy"],
    }
```

```
id2label = {i: label for i, label in enumerate(label_names)}
label2id = {v: k for k, v in id2label.items()}
```

هایپرپارامترهای مدل در عکس زیر قابل مشاهده هستند.

```python
from transformers import TrainingArguments

args = TrainingArguments(
    output_dir="ParsBERT_V3_ner_results",
    # evaluation_strategy="epoch",
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    learning_rate=2e-5,
    num_train_epochs=20,
    save_strategy="epoch",
    weight_decay=0.01,
    logging_steps=20,
    save_steps=50,
    logging_dir='ParsBERT_V3_ner_logs'
)
```

```python
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    tokenizer=tokenizer,
)
```

```python
trainer.train()
```

```
/home/user01/miniconda3/envs/mehrab2/lib/python3.10/site-packages/tra
ng: This implementation of AdamW is deprecated and will be removed in
ntation torch.optim.AdamW instead, or set `no_deprecation_warning=Tru
  warnings.warn(
```

[560/560 36:24, Epoch 20/20]

Step	Training Loss
20	1.539100
40	0.877900
60	0.665400
80	0.572400
100	0.511400
120	0.472600
140	0.406900
160	0.379400

نتایج مدل پس از ۲۰ ایپاک

```
trainer.evaluate()
```

[4/4 00:03]

```
{'eval_loss': 0.8843041658401489,
 'eval_precision': 0.40850277264325324,
 'eval_recall': 0.4682203389830508,
 'eval_f1': 0.4363277393879565,
 'eval_accuracy': 0.8177975058127246,
 'eval_runtime': 5.1568,
 'eval_samples_per_second': 10.86,
 'eval_steps_per_second': 0.776,
 'epoch': 20.0}
```

همچنین ما دیتاست را بر روی مدل ParsBert v۲ NER نیز با ۱۰ ایپاک ترین کردیم که نتایج بدست آمده را در زیر مشاهده می‌کنید.

[4/4 05:37]

```
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/v1.py:57: UndefinedMet
  _warn_prf(average, modifier, msg_start, len(result))

{'eval_loss': 1.1688835620880127,
 'eval_precision': 0.4036697247706422,
 'eval_recall': 0.4661016949152542,
 'eval_f1': 0.4326450344149459,
 'eval_accuracy': 0.810604251253881,
 'eval_runtime': 0.8666,
 'eval_samples_per_second': 64.623,
 'eval_steps_per_second': 4.616,
 'epoch': 10.0}
```

قابل مشاهده است علی‌رقم ترین شدن مدل ParsBert v۲ NER بر روی تسک NER باز هم مدل ParsBert v۳ اندکی بهتر عمل می‌کند.

بخش امتیازی – مدل NER