دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

استاد درس: دکتر محمدحسین رهبان

بهار ۱۴۰۰

گزارش فاز دوم پروژه یادگیری ماشین

درس یادگیری ماشین

نام و نام خانوادگی: امیر پورمند

شماره دانشجویی: ۹۹۲۱۰۲۵۹

آدرس ایمیل **pourmand1376@gmail.com**

فهرست مطالب

۱ مسئله واکاوی خوشه ها

۱.۱ روش های کاهش بعد

۲.۱ PCA

روش کاهش بعد PCA برای تبدیل داده ها به ابعاد پایین تر مورد استفاده قرار گرفته است که مشخص است با استفاده از کتابخانه sklearn از آن استفاده شده است. دقت کنیم که این تابع روی کل مجموعه train و test ترکیب شده با بازنمایی w۲v انجام شده است.

```
from sklearn.decomposition import PCA
pca=PCA(n_components=2)
pca_w2v=pca.fit_transform(X_w2v)
```

۳.۱ SVD

این روش نیز برای سادگی پیاده سازی انجام شده است و در واقع BOW کلمات را به SVD تبدیل کرده ایم و این برای مقاصد خوشه بندی قابل استفاده است. تمام ۳ الگوریتم استفاده شده با هر دوش روش مقایسه شده اند که البته روش w۲v بهتر عملکرده است.

```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=2, n_iter=7)
svd_bow=svd.fit_transform(X_bow)
```

۴.۱ روش های خوشه بندی

۱.۴.۱ روش KMeans

در ابتدای امر معرفی یک تابع که در روند کار برای plot کردن کلاسترهای مختلف به کار میرود را معرفی میکنم.

```
import matplotlib.pyplot as plt
def plot_scatter(X,pred):
    u_labels = np.unique(pred)
    for i in u_labels:
        plt.scatter(X[pred==i,0],X[pred==i,1],label=i)
    plt.legend()
    plt.show()
```

ما در اینجا از Kmeans با ۲ تا ۶ کلاستر استفاده کرده ایم و هر کدام رو plot کرده ایم ولی به علت این که نمیخوام کل این فایل پر از شکل شود و در واقع تکرار چیزهایی که قبلا در فایل نوت بوک بوده است به صرفا کد و توضیح ان بسنده میکنم. البته در اینجا از همان representation مبتنی بر word۲vec استفاده شده است ولی چون در کلاسترینگ اصولا جداکردن داده تست و ترین معنی ندارد، این داده ها را در واقع ترکیب کرده و ترکیب آنها استفاده میکنیم.

```
from sklearn.cluster import KMeans

for k in range(2,6):
    kmeans = KMeans(n_clusters=k)
    kmeans_label=kmeans.fit_predict(pca_w2v)
    plot_scatter(pca_w2v,kmeans_label)
```

Gaussian Mixture Model ۲.۴.۱

این روش نیز به گفته سوال پیاده سازی شده است که البته نتایج بهتری نسبت به روش kmeans دارد. اگر در شکل هم نگاه کنیم نتایجش قابل قبول است.

```
from sklearn.mixture import GaussianMixture

for k in range(2,6):
    gm = GaussianMixture(n_components=k)
    gm_pred=gm.fit_predict(pca_w2v)
    plot_scatter(pca_w2v,gm_pred)
```

Agglomerative Clustering ۳.۴.۱

با توجه به این که در منابع مختلفی این روش را دیده بودم فکر میکردم خیلی خیلی خوب عمل خواهد کرد که متاسفانه در اینجا عملکرد خیلی خوبی نداشت. البته در جدول sklearn نیز بیان شده است که برای داده های با همین مشخصات خوب خواهد بود ولی خوب نبود! و چون در کولب اگر با کل داده ها ترین میکردم کرش میکرد مجبور شدم صرفا سی هزار داده را انتخاب و از بین آنها خوشه بندی را انجام دهم. البته چند روش دیگر را هم پیدا کردم و آنها نیز همگی کرش کردند. (رم کم میآوردند.)

```
from sklearn.cluster import AgglomerativeClustering

max_data= 30000
for k in range(2,6):
    agg = AgglomerativeClustering(n_clusters=k)
    agg_pred=agg.fit_predict(pca_w2v[:max_data])
    plot_scatter(pca_w2v[:max_data],agg_pred)
```

۵.۱ مقایسه روش های خوشه بندی با ۲ دسته

برای مقایسه روش های مختلف خوشه بندی از تابع زیر استفاده کردم که در واقع ۵ معیار مختلف برای خوشه بندی است.

```
from sklearn import metrics

def get_analysis(name,
true_label,predicted_label):
    print(' Measure V', name,
    ':', metrics.v_measure_score(true_label,predicted_label))
    print(' Measure RandScore Adjusted', name,
     ':', metrics.adjusted_rand_score(true_label,predicted_label))
    print(' Information Mutual Adjusted', name,
    ':', metrics.adjusted_mutual_info_score(true_label,predicted_label))
    print('Homogenity', name,
     ':', metrics.homogeneity_score(true_label,predicted_label))
    print('-'*30)
```

در اینجا نیز با توجه به معیارهای بدست آمده که همگی مشخص هستند بهترین نتیجه از آن GMM و سپس KMeans و سپس agglomorative است .

```
V Measure  kmeans : 03865260111674556.0
Adjusted RandScore Measure  kmeans : 052532117323756226.0
Adjusted Mutual Information  kmeans : 038637148304102975.0
Homogenity kmeans : 038548295065080805.0
------------------------------
V Measure  gm : 04585216408127864.0
Adjusted RandScore Measure  gm : 06102151128817464.0
Adjusted Mutual Information  gm : 045836750679784814.0
Homogenity gm : 045502049103971876.0
------------------------------
V Measure  agg : 027294842860872137.0
Adjusted RandScore Measure  agg : 015243361512629535.0
Adjusted Mutual Information  agg : 02726576198577149.0
Homogenity agg : 021955360291170237.0
------------------------------
```

۶.۱ مقایسه مفهوم دسته های مختلف

در اینجا بنده ۳ دسته را در نظر گرفتم و سعی کردم هر کدام را بصورت شهودی تحلیل کنم و به این نتیجه رسیدم که در واقع دسته اول نظرات خیلی منفی را در بردارد و دسته دوم اکثرا نظرات خیلی مثبت را دارد و در واقع دسته سوم نیز نظراتی را مطرح میکند که خوب هستند ولی خیلی زیاد تعریف نمیکنند و میتوان گفت به نوعی به هر دو دسته میتوانند تعلق داشته باشند.

```
gm = GaussianMixture(n_components=3)
gm_pred=gm.fit_predict(pca_w2v)
for i in range(3):
    print(list(dataset[gm_pred==i][2:3]['sentiment']))
for i in range(3):
    print(list(dataset[gm_pred==i][2:3]['comment']))
```

۲ مسئله Fine-tuning

۱.۲ پیاده سازی MLP روی دیتاست

در اینجا بنده دوباره تمام پیش پردازش هایی که در مرحله قبل انجام شده بود را انجام دادم و از روش TF-IDF برای تبدیل کلمات به بردار استفاده کردم که قبلا توضیح آن در گزارش فاز دوم آماده است.
یک GridSearch نیز برای تنظیم دقیق تر پارامترها در cross-validation انجام شده است که مشخص است که بهترین مدل، آن مدلی است که از ۱ لایه استفاده کرده است که در ان ۹۰ نورون وجود دارد. سپس همین مدل فیت شده است و دقت ۸۸ درصدی را از آن خود کرده است.

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
grid_params = {
    'hidden_layer_sizes':[(250),(100),(90),(40,10),(50,10)]
}
mlp = MLPClassifier(learning_rate='adaptive',solver='adam',max_iter=1000)
mlp_cv = GridSearchCV(estimator=mlp,param_grid=grid_params,cv=2)
mlp_cv.fit(X_train_small_tfidf,y_train_small)
mlp_prediction=mlp_cv.predict(X_test_small_tfidf)
print_confusion_matrix(y_test_small,mlp_prediction,' MLP TFIDF:')
display(pd.DataFrame( mlp_cv.cv_results_))
```

۲.۲ fine-tune کردن MLP با استفاده از داده های جدید

میتوان گفت یکی از بهترین مدل ها در فاز قبل مدل MLP است که در اینجا من برای بحث fine-tuning استفاده کرده ام و توانسته است دقت ۸۹ درصدی بگیرد و ۱ درصد دقت را افزایش دهد که به نسبت خوب است. برای این کار از پارامتر warm_start در مدل MLP استفاده کرده ام. البته همان طور که مشخص است از مدل TFIDF ای که در قبل fit شده است استفاده شده و اینجا صرفا transform انجام شده است.

```
X_train_small_tfidf_olddata=vectorizer_tfidf.transform(X_train_small)
X_test_small_tfidf_olddata = vectorizer_tfidf.transform(X_test_small)

mlp_best = MLPClassifier(warm_start=True)
mlp_best.fit(X_train_small_tfidf_olddata,y_train_small)
mlp_prediction=mlp_best.predict(X_test_small_tfidf_olddata)
print_confusion_matrix(y_test_small,mlp_prediction,' MLP TFIDF:')
```

۳ مدل های آموزش داده شده و لینک ها

در این لینک میتوان تمام مدل های آموزش داده شده و البته برخی representation ها که ذخیره شده اند را پیدا کرد. در این لینک هم در واقع فایل کولب تمرین است که در صورت لزوم قابل بررسی است.