دانشکده مهندسی کامپیوتر
دانشگاه صنعتی شریف

استاد درس: دکتر محمدحسین رهبان

بهار ۱۴۰۰

گزارش فاز اول پروژه یادگیری ماشین
درس یادگیری ماشین

نام و نام خانوادگی: امیر پورمند

شماره دانشجویی: ۹۹۲۱۰۲۵۹

آدرس ایمیل **pourmand1376@gmail.com**

فهرست مطالب

۱ پیش پردازش

۱.۱ توابع پیش پردازش

با توجه به نتایج بدست آمده عملا پیش پردازش داده ها تاثیر چندان زیادی در نتیجه ندارد که بر خلاف انتظار اولیه بنده است. با این تفاسیر توضیح برخی کدهای بنده خالی از لطف نیست.

در این تابع تمام حرف های غیرالفبا با یک رجکس ساده حذف می شوند.

```
def remove_all_non_alphabetic(text):
  return re.sub('[^A-Za-z]',' ',text)
```

در این تابع تگ های html که ممکن است در متن وجود داشته باشند همگی حذف میشوند.

```
def strip_html_tags(text):
    text""" from tags html """remove
    soup = BeautifulSoup(text, "html.parser")
    stripped_text = soup.get_text(separator=" ")
    return stripped_text
```

در اینجا کلمات stop word انگلیسی حذف میشوند که در واقع از یک دیکشنری انگلیسی با استفاده از کتابخانه استخراج شده اند.

```
stop_words = set(stopwords.words('english'))
def remove_stop_words(token):
  return [item for item in token if item not in stop_words]
```

بنده از بین دو تابع stemmatization و lemmatization در واقع lemmatization را انتخاب کردم. علت انتخاب هم بود که به نظرم منطقی تر است.

```
lemma = WordNetLemmatizer()
def lemmatization(token):
  return [lemma.lemmatize(word=w,pos='v') for w in token]
```

در اینجا هم صرفا یک تابع برای حذف کلمات کوچکتر از ۲ حرف نوشتم که واقعا ساده است.

```
def clean_length(token):
  return [item for item in token if len(item)>2]
```

برای حذف نقطه و ویرگول و کاما و بقیه علائم نگارشی نوشته شده است که البته تاثیر زیادی هم نداشت.

```
def punctuation_removal(text):
    return re.sub(r',\:\;\"][\.\?\!\', '', text)
```

این تابع هم برای جوین کردن متن tokenize شده مورد استفاده است.

```
def text_merge(token):
  return ' '.join([i for i in token if not i.isdigit()])
```

در نهایت هم با استفاده از تابع زیر پیش پردازش کلیه متن ها را انجام داده ام. لازم به ذکر است که ۳ متغیر مستقل از هم برای نگه داشتن پیش پردازش های مختلف در فایل وجود دارد که با عدد انتهایی صفر و یک و دو نشان داده شده اند. یعنی X_train_0 به معنای داده پیش پردازش نشده است. X_train_1 به معنای داده ای هست که مرحله اول پیش پردازش روی آن انجام شده است و نهایتا X_train_2 داده ای هست که تمام پیش پردازش های مدنظر بنده رو آن اعمال شده است.

```
def process_level1(data):
    return (data.apply(str.lower)
                .apply(remove_all_non_alphabetic)
                .apply(word_tokenize)
                .apply(text_merge))

def process_level2(data):
    return (data.apply(str.lower)
        .apply(contractions.fix)
        .apply(strip_html_tags)
        .apply(remove_accented_chars)
        .apply(remove_all_non_alphabetic)
        .apply(word_tokenize)
        .apply(remove_stop_words)
        .apply(lemmatization)
        .apply(clean_length)
        .apply(text_merge))
```

۲.۱ پیاده سازی BOW

برای پیاده سازی روش Bag of Wordsکه در این فایل اختصارا به BOW میشناسم از فایل CountVectorizer کتابخانه sklearn استفاده شده است. نکته مهم در این کتابخانه دو پارامتر max_df و min_df است که برای این که همه کلمات تبدیل نشوند استفاده شده است. برای مثال بنده میخواستم کلماتی که در زیر ۱ درصد کامنت ها موجود است و البته در بالای ۴۰ درصد کامنت ها وجود دارد حذف شود زیرا این کلمات یا خیلی نادر هستند و یا خیلی پرتکرار هستند و تفاوت زیادی در پیش بینی ایجاد نمیکنند.

در ضمن در روش SVM بدون وجود این پارامترها دقت بسیار بد بود (در حد ۵۰ درصد) ولی با این روش نزدیک ۸۸ درصد جواب داده است.

```
def convert_to_BOW(train,test):
    vectorizer = CountVectorizer(max_df=4.0,min_df=01.0,lowercase=False)
    X_train_transformed = vectorizer.fit_transform(train)
    X_test_transformed = vectorizer.transform(test)
    return X_train_transformed,X_test_transformed
```

۳.۱ نتیجه LR

برای رگرسیون خطی از کد زیر استفاده کردم و پارامترها را پاس داده ام که البته همان طور که در شکل ها مشخص است تغییر به خصوصی در پیش پردازش ها بوجود نیامده است و همه دقت ها و معیارها ۸۸ درصد است و تقریبا در همگی برابر است.

```
from sklearn.linear_model import LogisticRegression
def regression(X_train,X_test,y_train,**kwarg):
    clf = LogisticRegression(**kwarg).fit(X_train, y_train)
    return clf.predict(X_test),clf
```

۴.۱ نتیجه KNN

در مورد KNN اما موضوع قدری متفاوت است و دقت ها به ترتیب در هر مرحله پیش پردازش ۲ درصد زیاد شده اند که به نسبت بسیار خوب است. (دقت ها به ترتیب ۶۳، ۶۵ و ۶۷ برای داده های بدون پیش پردازش و داده های با پیش پردازش اولیه و داده های با پیش پردازش کامل هستند)

در پیاده سازی این تابع نیز همانند بقیه از کتابخانه آماده sklearn استفاده شده است.

```
from sklearn.neighbors import KNeighborsClassifier
def knn(X_train,X_test,y_train):
    neigh = KNeighborsClassifier(n_neighbors=5)
    neigh.fit(X_train, y_train)
    return neigh.predict(X_test),neigh
```

۵.۱ نتیجه SVM

در مورد SVM مانند روش LR دقت ها در هر ۳ پیش پردازش ۸۶ درصد است. این روش بیشترین زمان را برای ترین کردن به خود اختصاص داد.

```
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
def svm(X_train,X_test,y_train):
    clf = make_pipeline( SVC(gamma='auto'))
    clf.fit(X_train, y_train)
    return clf.predict(X_test),clf
```

۲ روش های مختلف بردار سازی

۱.۲ معرفی

به طور کلی در اینجا من از ۳ روش بردار سازی استفاده کردم که عبارتند از Bag of Words ، Word to Vector و TF-IDF.
روش BOW را که در فاز قبل توضیح داده ام. در اینجا روش W۲v را توضیح میدهم و tf-idf را در قسمت بعدی به همراه MLP توضیح میدهم.
در این روش هر کلمه به یک بردار تبدیل میشود که لزوما با کلماتی که شباهت زیادی به هم دارند بردارهای نزدیکی دارند و به سادگی میتوان ارتباط بین کلمات مختلف را مشاهده کرد.

```
import gensim.models
w2v = gensim.models.Word2Vec( [row.split() for row in X_train_2],
                              min_count=50,
                             window=10,
                             size=300)
```

برای مثال در اینجا بنده یک مثال برای کلمه king یا همان پادشاه زده ام که ببینیم کدام کلمات به آن نزدیک تر هستند.

```
w2v.most_similar('king')
output:
[('lion', 7712827324867249.0),
 ('stephen', 7464975118637085.0),
 ('solomon', 7280431985855103.0),
 ('lord', 706125020980835.0),
 ('legend', 6681618094444275.0),
 ('princess', 6676332950592041.0),
 ('kings', 6650164127349854.0),
 ('rice', 6360715627670288.0),
 ('queen', 6311061978340149.0),
 ('immortal', 6231356859207153.0)]
```

برای تبدیل کردن جملات به بردار هم از تابع زیر استفاده کرده ام

```
def document_vector(doc):
    doc = [word for word in doc.split() if word in w2v.wv.vocab]
    return np.mean(w2v[doc], axis=0)
```

۲.۲ مقایسه روش های بردارهای سازی مختلف

جدول ۱: مقایسه روش های مختلف بردار سازی

روش های مختلف	LR	KNN	SVM
BOW	۸۷	۶۷	۸۷
W۲V	۸۷	۸۲	۸۷

همان طور که مشخص است برای دو روش LR و SVM روش بردار سازی هیچ تفاوتی ندارند اما برای KNN تفاوت مشهود است. البته میتوان گفت در کل برای این تسک دو روش LR و SVM بهتر هستند و البته LR به نظر بنده بهتر از SVM میتواند باشد زیرا زمان آموزش بسیار بسیار کمتری نیاز دارد.

۳ پیاده سازی مدل MLP

۱.۳ توضیح روش TF-IDF

در این روش نیز با استفاده از کتابخانه sklearn کلمات شمرده میشوند و نسبت آنها به کل کلمات با توجه به فرمولی در نظر گرفته میشود که با توجه به این که همین معیار در موتورهای جستجوگر نیز استفاده میشود فکر کردم بد نیست آن را پیاده سازی کنم. البته باز هم به علت زیاد بودن کلمات اضافی ، کلماتی که زیر ۱ درصد تکرار شده اند و انهایی که بالای ۵۰ درصد تکرار شده اند حذف شده اند تا نتیجه بهتری بتوان گرفت.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer( min_df=01.0,max_df=5.0)
X_train_2_tfidf=vectorizer.fit_transform(X_train_2)
X_test_2_tfidf = vectorizer.transform(X_test_2)
```

۲.۳ توضیح روش MLP

اولا در اینجا از cross-validation استفاده شده است تا بتوان نتیجه قابل اتکا تری گرفت و همان طور که مشخص است مدل ها با تعداد لایه های مختلف با fold=۲ تست شده اند .

```
grid = {
    'hidden_layer_sizes':[(80),(70,),(40,10),(90)],
}
mlp = MLPClassifier(learning_rate='adaptive',solver='adam')
mlp_cv = GridSearchCV (estimator=mlp,param_grid=grid,cv=2)
```

سپس خود مدل ترین شده است که دقت ۸۷ درصد را روی داده های تست بدست آورده است.

```
mlp_cv.fit(X_train_2_tfidf,y_train)

mlp_prediction=mlp_cv.predict(X_test_2_tfidf)
print_confusion_matrix(y_test,mlp_prediction,' MLP TFIDF:')
```

۴ مدل های آموزش داده شده

در این لینک تمام مدل های اموزش دیده شده قرار دارد. البته قبلا فایل ها با ایمیل های گفته شده در اشتراک گذاشته شده است. این فولدر علاوه بر چیزهای خواسته شده فایل داده های پردازش شده در هر مرحله را نیز در بردارد که به سادگی قابل تست و بررسی است.

در این لینک نیز فایل کولب برای این پروژه قرار دارد که میتواند در صورت نیاز بررسی شود.