

✓ In Class Assignment: Sentiment Analysis

Name: Dallin Moore

DATA 5420/6420

In this in-class assignment we will examine the sentiment of movie reviews using both unsupervised lexicon-based modeling and through supervised classification. We will then leverage tools to interpret the decision of our sentiment analysis model to determine the words and topics associated with positive and negative sentiments.

We begin, as always, by importing our dependencies:

```
import pandas as pd
import numpy as np
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer

import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
nltk.download('punkt')
nltk.download('stopwords')

import sklearn
from sklearn.metrics import classification_report
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

import matplotlib.pyplot as plt
%matplotlib inline

np.set_printoptions(precision=2, linewidth=80)

import warnings
warnings.filterwarnings("ignore")

[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data] Package vader_lexicon is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

Let's examine the vader dictionary a bit. What do these scores represent, and how are they interpreted?

The output below shows specific words and their associated score. The score has zero as neutral and its magnitude in the positive/negative direction relate to the positive/negative sentiment that the word has.

```
sid = SentimentIntensityAnalyzer()
vader_lexicon = sid.lexicon

# Display a preview of the lexicon
dict(list(vader_lexicon.items())[1000:1035])

{'beauties': 2.4,
 'beautification': 1.9,
 'beautifications': 2.4,
 'beautified': 2.1,
 'beautifier': 1.7,
 'beautifiers': 1.7,
 'beautifies': 1.8,
 'beautiful': 2.9,
 'beautifuler': 2.1,
 'beautifulest': 2.6,
 'beautifully': 2.7,
 'beautifulness': 2.6,
```

```

'beautify': 2.3,
'beautifying': 2.3,
'beauts': 1.7,
'beauty': 2.8,
'belittle': -1.9,
'belittled': -2.0,
'beloved': 2.3,
'benefic': 1.4,
'benefice': 0.4,
'beneficed': 1.1,
'beneficence': 2.8,
'beneficences': 1.5,
'beneficent': 2.3,
'beneficently': 2.2,
'benefices': 1.1,
'beneficial': 1.9,
'beneficially': 2.4,
'beneficialness': 1.7,
'beneficiaries': 1.8,
'beneficiary': 2.1,
'beneficiate': 1.0,
'beneficiation': 0.4,
'benefit': 2.0}



```

Load and Normalize the Dataset

```

df = pd.read_csv('/content/movie_reviews-1.csv')
df.head()

```

	review	sentiment	
0	One of the other reviewers has mentioned that ...	positive	
1	A wonderful little production. The...	positive	
2	I thought this was a wonderful way to spend ti...	positive	
3	Basically there's a family where a little boy ...	negative	
4	Petter Mattei's "Love in the Time of Money" is...	positive	

Next steps: [View recommended plots](#)

Sentiment Distribution

```

# @title Sentiment Distribution

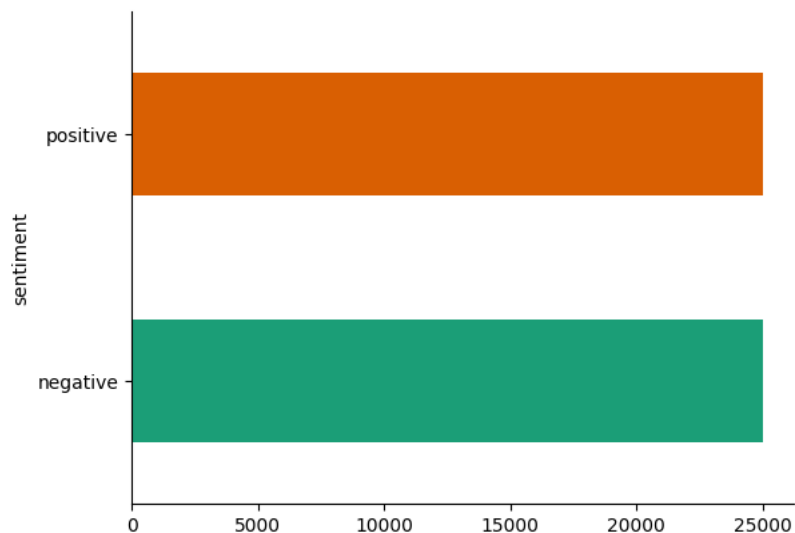
```

```

from matplotlib import pyplot as plt
import seaborn as sns
df.groupby('sentiment').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right']].set_visible(False)
df['sentiment'].value_counts()

```

```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```



```
def clean_text(text):
    text = text.lower() # lowercase
    text = re.sub(r'\d+', '', text) # substituting digits with nothing
    text = re.sub(r'[^\w\s]', '', text) # removes anything that isn't a-z character
    tokens = nltk.word_tokenize(text)
    tokens = [word for word in tokens if word not in stopwords.words('english')]
    text = ' '.join(tokens)
    return text
```

which typical normalization step did we skip

```
df['clean_review'] = df['review'].apply(clean_text)
df.head()
```

```
df = pd.read_csv('/content/cleaned_movies.csv')
df.head()
```

	Unnamed: 0	review	sentiment	clean_review
0	0	One of the other reviewers has mentioned that ...	positive	one reviewers mentioned watching oz episode yo...
1	1	A wonderful little production. The...	positive	wonderful little production br br filming tech...
2	2	I thought this was a wonderful way to spend ti...	positive	thought wonderful way spend time hot summer we...

Next steps: ☒ [View recommended plots](#)

```
reviews = np.array(df['clean_review'])
sentiments = np.array(df['sentiment'])

test_reviews = reviews[9500:]
test_sentiments = sentiments[9500:]
sample_review_ids = [9511,9622,9880]
```

✓ Sentiment Analysis with VADER

Create a Scoring Function:

```
# Add in comments
def analyze_sentiment_vader_lexicon(review,
                                   threshold=0.1,
                                   verbose=False): # for additional stats
    analyzer = SentimentIntensityAnalyzer()
    scores = analyzer.polarity_scores(review) # score the words

    agg_score = scores['compound'] # overall score
    final_sentiment = 'positive' if agg_score >= threshold\
        else 'negative'

    if verbose:
        positive = str(round(scores['pos'], 2)*100)+'%' # percent of words positive
        final = round(agg_score, 2)
        negative = str(round(scores['neg'], 2)*100)+'%'
        neutral = str(round(scores['neu'], 2)*100)+'%'
        sentiment_frame = pd.DataFrame([[final_sentiment, final, positive,
                                         negative, neutral]],
                                       columns=pd.MultiIndex(levels=[['SENTIMENT STATS:'],
                                                                    ['Predicted Sentiment', 'Polarity Score',
                                                                     'Positive', 'Negative', 'Neutral']],
                                                             codes=[[0,0,0,0,0],[0,1,2,3,4]]))

        print(sentiment_frame)

    return final_sentiment
```


✓ Predict Sentiment of Sample Reviews

```
for reviews, sentiments in zip(reviews[sample_review_ids], sentiments[sample_review_ids]):
    print('REVIEW:', reviews)
    print('Actual Sentiment:', sentiments)
    pred = analyze_sentiment_vader_lexicon(reviews, threshold = 0.4, verbose = True)
    print('-'*60)
```

REVIEW: scientists behaved way hg wells confident would future history wouldnt quite turned way things come almost years past point well
Actual Sentiment: positive
SENTIMENT STATS:
Predicted Sentiment Polarity Score Positive Negative Neutral
0 negative 0.32 20.0% 17.0% 64.0%

REVIEW: found good movie pass time chance historical value portrayal cleopatra reminded cheap soap operabr br twist facts funny gave bir
Actual Sentiment: negative
SENTIMENT STATS:
Predicted Sentiment Polarity Score Positive Negative Neutral
0 negative -0.59 18.0% 20.0% 61.0%

REVIEW: always loved film musicstory action especially love opening closing film music stayed throughout years wwi plane battles great c
Actual Sentiment: positive
SENTIMENT STATS:
Predicted Sentiment Polarity Score Positive Negative Neutral
0 positive 0.99 42.0% 3.0% 55.0000000000001%



```
df['review'].loc[9880]
```

'I always loved this film. The music,story and the action. I especially love the openin
g and closing of the film. The music stayed with me throughout the years. The WWI plane
battles were great and the comedy is typical Blake Edwards. Slaptick is his forte' afte
r all. Julie's singing is amazing and keeps me glued to the screen. The sets and the sc
enes are wonderful. The characters are appealing. I loved the scene with the wounded so
ldiers and Julie's singing to them I wish she sang to me in Vietnam I also enjoyed th

What impact does the threshold value have on the classification of sentiments within reviews?

How strong of a sentiment score is needed to classify the text as positive. The closer it is to -3 (for this specific model), the lower the positive sentiment needs to be for it to be classified as such.

Do you agree with all of the ratings? If not, what words or phrases might be throwing off the sentiment score?

not necessarily. The high amount of neutral words is causing for lower scores. I think that for this dataset specifically there is less indication of positive or negative sentiment. I was surprised rating 9622 was correctly classified because of the sarcasm present (it starts by saying it was a

good movie), and rating 9511 is harder to identify by looking at it, but it also seems positive. Rating 9880 was by far the most positive case (with an unsurprising score of .99).

✓ Next let's predict sentiment via machine learning, using several classification algorithms...

Build a training and test set

✓ Feature Engineering

```
x = df['clean_review']
y = df['sentiment']

train_reviews, test_reviews, train_sentiments, test_sentiments = train_test_split(x, y, test_size=0.2, random_state=42)
# consider unigrams and bigrams (one and two word phrases)
tv = TfidfVectorizer(use_idf=True, min_df=5, max_df=0.6, ngram_range=(1,2), sublinear_tf=True)
tv_train_features = tv.fit_transform(train_reviews)
tv_test_features = tv.transform(test_reviews)

tv_train_features.shape, tv_test_features.shape

((8000, 30582), (2000, 30582))
```

✓ Model Training, Prediction, and Performance Eval

```
lr = LogisticRegression(penalty='l2', max_iter=100, C=1) # linear model
svm = SGDClassifier(loss='hinge', max_iter=100) # and non-linear
```

✓ Logistic Regression with TF-IDF Features

```
def train_predict_model(classifier,
                        train_features, train_labels,
                        test_features, test_labels):
    classifier.fit(train_features, train_labels)
    predictions = classifier.predict(test_features)
    return predictions

lr_tfidf_predictions = train_predict_model(classifier=lr,
                                          train_features=tv_train_features, train_labels=train_sentiments,
                                          test_features=tv_test_features, test_labels=test_sentiments)

print(classification_report(test_sentiments, lr_tfidf_predictions))
```

	precision	recall	f1-score	support
negative	0.91	0.87	0.89	996
positive	0.87	0.91	0.89	1004
accuracy			0.89	2000
macro avg	0.89	0.89	0.89	2000
weighted avg	0.89	0.89	0.89	2000

```
svm_tfidf_predictions = train_predict_model(classifier=svm,
                                          train_features=tv_train_features, train_labels=train_sentiments,
                                          test_features=tv_test_features, test_labels=test_sentiments)

print(classification_report(test_sentiments, svm_tfidf_predictions))
```

	precision	recall	f1-score	support
negative	0.90	0.86	0.88	996
positive	0.87	0.90	0.89	1004
accuracy			0.88	2000
macro avg	0.88	0.88	0.88	2000
weighted avg	0.88	0.88	0.88	2000

Which metric would be most appropriate to use in this case and why?

The linear regression model was slightly more accurate for both precision and recall and therefor should be used. In this case, since we don't know the use case or whether false-positives or false-negatives are more costly, simply looking at the f1-scores reveals which model is better (although using recall/precision would give the same result).

✓ Interpretation Time!

First let's examine predicted probabilities to see how confident our model was in its predictions...

```
model = lr.fit(tv_train_features, train_sentiments)

predicted_probab = list(zip(*model.predict_proba(tv_test_features)))[0]

predictions_df = pd.DataFrame()
predictions_df['Reviews'] = test_reviews
predictions_df['True Sentiment'] = test_sentiments
predictions_df['Predicted Sentiment'] = lr_tfidf_predictions
predictions_df['Probability_Negative'] = predicted_probab

predictions_df
```

index:	Reviews:	×
<input type="text"/> to <input type="text"/>	<input type="text"/>	
True Sentiment:	Predicted Sentiment:	
<input type="text"/>	<input type="text"/>	
Probability_Negative:		
.45 <input type="text"/> to <input type="text"/>		
Search by all fields:		
<input type="text"/>		

Index	Reviews	True Sentiment	Predicted Sentiment	Probability_Negative ▲
483	one word describe movie weird recorded movie one day japanese animation old thought would interesting well movie young boy travels universe get metal body seek revenge way meets colorful characters must ultimately decide wants body strange fan animationsciencefiction might want check	positive	positive	0.45000319995869265
4540	although allegedly autobiographical movie demonstrates little insight protagonists psychology resulting flat fragmented characterization well largerscale historical processes hope either learning something new improving understanding contemporary iran remained unfulfilled instead found sensibilities somewhat dulled succession bearded islamic villains replaced taunting torturing killing wantonly victimized prototypical middleclass iranian whose western cultural sympathies patent whose exoneration movie quite blatantly seeks deeper understanding movie seem demonstrate massmedia market serves nourish prevalent occidental folkideologies ie crowd pleaser br br redeemed movie outright boring creative animation genuinely minimalistic imagery nevertheless always kept screen rich expressive unambiguous small fete give stars	negative	positive	0.45007961208009994
7684	another winner era love much comedic value give viewing days corman never lets films take seriously low budgets recipe good watching sure ed nelson competent actor got started corman well many favorites show superman many westerns day costume pretty bad sound alien speaking well reverb little thats beauty film making love looking perfection digging action comes thru films fun time even better mstk version	negative	positive	0.4511857087265545
	viewing please make sure seen night living dead might well best			

Next steps: [🔍 View recommended plots](#)

Sort the predictions_df by predicted probability to find movies with high, moderate and low probabilities to examine a) whether the predictions were correct and b) try to identify what sorts of words or phrases led to high or low probabilities of a negative sentiment

At the top and bottom of the spectrum the model was really accurate at predicting positive/negative. However, in the middle it was less accurate and even as a human reading the text it is somewhat difficult to tell whether the prediction is positive or negative (it's not black and white).

The word 'love' seems very powerful indicator of positive sentiment.

Looking at rating 9186, the model was not very certain that it was positive (0.458 probability negative) but as a reader it was very clearly negative because it ended saying the movie 'severely disappoints.' I don't know where the positive sentiment came from in this case, perhaps it

was just mostly neutral.

▼ Feature Importance Plot:

```
# Add in comments
def plot_top_feature_importance_updated(coefficients, feature_names, title, top_n=20):
    """Helper function to plot top feature importance using only matplotlib"""
    coef_df = pd.DataFrame({'feature': feature_names, 'coefficient': coefficients})
    coef_df = coef_df.reindex(coef_df.coefficient.abs().sort_values(ascending=False).index)

    plt.figure(figsize=(10, 6))

# use matplotlib to create a bar chart to display the topp features and their coefficients from the lr

    top_features = coef_df.head(top_n)
    plt.barh(top_features['feature'], top_features['coefficient'], align='center', color='skyblue')
    plt.title(title)
    plt.xlabel('Coefficient Value')
    plt.ylabel('Feature')
    plt.gca().invert_yaxis()
    plt.show()

# only use 5000 observations
subset_size = 5000
subset_reviews = train_reviews[:subset_size]
subset_sentiments = train_sentiments[:subset_size]

tv = TfidfVectorizer(use_idf=True, min_df=5, max_df=0.6, ngram_range=(1,2),
                    sublinear_tf=True, max_features=5000)
tv_train_features_subset = tv.fit_transform(subset_reviews)

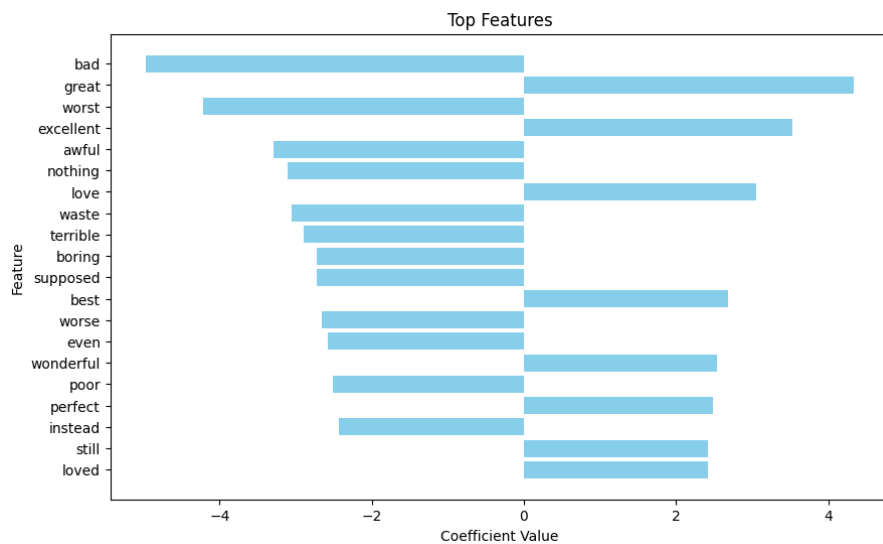
# rerun lr on subset

lr = LogisticRegression(max_iter=1000)
lr.fit(tv_train_features_subset, subset_sentiments)

# extract the sentiment
predicted_sentiments_subset = lr.predict(tv_train_features_subset)

# extract the coefficients and the vocab name
coefficients_subset = lr.coef_[0]
feature_names_subset = tv.get_feature_names_out()

plot_top_feature_importance_updated(coefficients_subset, feature_names_subset, 'Top Features', top_n=20)
```

▼ Apply to unlabeled data:

```
# Toy sample movie reviews
sample_reviews = [
    "The film was a masterpiece, with brilliant acting and a captivating storyline.",
    "I didn't enjoy the movie at all. The plot was dull and the characters were uninteresting.",
    "A decent watch. Some parts were great, while others were just okay.",
    "Absolutely loved it! The cinematography was top-notch and the performances were unforgettable.",
    "It felt like a waste of time. The jokes were flat and the pacing was off.",
    "What a waste of my time! It was worse than I thought it would be but my wife loved it.",
    "It was nothing special. I'll only watch it again if I had to.",
    "This movie is different. I've never stayed so engaged during a movie!",
    "It was definitely interesting. I'm curious if it will get a sequel, it couldn't be worse than this!"
]

# Clean the sample reviews
cleaned_sample_reviews = [clean_text(review) for review in sample_reviews]

# Transform the cleaned reviews using the previously trained TF-IDF vectorizer
tv_sample_features = tv.transform(cleaned_sample_reviews)

# Predict sentiments using the trained logistic regression model
predicted_sample_sentiments = lr.predict(tv_sample_features)

predicted_sample_sentiments

array(['positive', 'negative', 'positive', 'positive', 'negative', 'negative',
      'negative', 'positive', 'negative'], dtype=object)
```

▼ [Hugging Face](#) Pretrained Models**

```
!pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.35.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.20.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.25.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2023.12.25)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.31.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.15.2)
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.4.2)
```

Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.2)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformers) (2024.2.2)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformers) (4.6.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2024.2.2)

```
from transformers import pipeline
```

```
classifier = pipeline('sentiment-analysis')
```