Building a Sentiment Classifier using Scikit-Learn

Acknowledgement: This is derived from https://towardsdatascience.com/building-a-sentiment-classifier-using-scikit-learn-54c8e7c5d2f0 (https://towardsdatascience.com/building-a-sentiment-classifier-using-scikit-learn-54c8e7c5d2f0).

Image by AbsolutVision @ pixabay.com (https://pixabay.com/ro/photos/smiley-emoticon-furie-sup%C4%83rat-2979107/)

> **Sentiment analysis**, an important area in Natural Language Processing, is the process of automatically detecting affective states of text. Sentiment analysis is widely applied to voice-of-customer materials such as product reviews in online shopping websites like Amazon, movie reviews or social media. It can be just a basic task of classifying the polarity of a text as being positive/negative or it can go beyond polarity, looking at emotional states such as "happy", "angry", etc.

Here we will build a classifier that is able to distinguish movie reviews as being either positive or negative. For that, we will use Large Movie Review Dataset v1.0 (http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz)[2] of IMDB movie reviews. This dataset contains 50,000 movie reviews divided evenly into 25k train and 25k test. The labels are balanced between the two classes (positive and negative). Reviews with a score <= 4 out of 10 are labeled negative and those with score >= 7 out of 10 are labeled positive. Neutral reviews are not included in the labeled data. This dataset also contains unlabeled reviews for unsupervised learning; we will not use them here. There are no more than 30 reviews for a particular movie because the ratings of the same movie tend to be correlated. All reviews for a given movie are either in train or test set but not in both, in order to avoid test accuracy gain by memorizing movie-specific terms.

Data preprocessing

After the dataset has been downloaded and extracted from archive we have to transform it into a more suitable form for feeding it into a machine learning model for training. We will start by combining all review data into 2 pandas Data Frames representing the train and test datasets, and then saving them as csv files: *imdb_train.csv* and *imdb_test.csv*.

The Data Frames will have the following form:

text	label
review1	0
review2	1
review3	1
.....	...
reviewN	0

where:

- review1, review2, ... = the actual text of movie review
- 0 = negative review
- 1 = positive review

But machine learnng algorithms work only with numerical values. We can't just input the text itself into a machine learning model and have it learn from that. We have to, somehow, represent the text by numbers or vectors of numbers. One way of doing this is by using the **Bag-of-words** model[3], in which a piece of text(often called a **document**) is represented by a vector of the counts of words from a vocabulary in that document. This model doesn't take into account grammar rules or word ordering; all it considers is the frequency of words. If we use the counts of each word independently we name this representation a **unigram**. In general, in a **n-gram** we take into account the counts of each combination of n words from the vocabulary that appears in a given document.

For example, consider these two documents:

```
d1: "I am learning"

d2: "Machine learning is cool"
```

The vocabulary of all words encountered in these two sentences is:

```
v: [ I, am, learning, machine, is, cool ]
```

The unigram representations of d1 and d2:

unigram(d1)	I	am	learning	machine	is	cool
	1	1	1	0	0	0

unigram(d2)	I	am	learning	machine	is	cool
	0	0	1	1	1	1

And, the bigrams of d1 and d2 are:

bigram(d1)	I I	I am	I learning	...	machine am	machine learning	...	cool is	cool cool
	0	1	0	...	0	0	...	0	0

bigram(d2)	I I	I am	I learning	...	machine am	machine learning	...	cool is	cool cool
	0	0	0	...	0	1	...	0	0

Often, we can achieve slightly better results if instead of counts of words we use something called **term frequency times inverse document frequency** (or **tf-idf**). Maybe it sounds complicated, but it is not. Bear with me, I will explain this. The intuition behind this is the following. So, what's the problem of using just the frequency of terms inside a document? Although some terms may have a high frequency inside documents they may not be so relevant for describing a given document in which they appear. That's because those terms may also have a high frequency across the collection of all documents. For example, a collection of movie reviews may have terms specific to movies/cinematography that are present in almost all documents(they have a high **document frequency**). So, when we encounter those terms in a document this doesn't tell much about whether it is a positive or negative review. We need a way of relating **term frequency** (how frequent a term is inside a document) to **document frequency** (how frequent a term is across the whole collection of documents). That is:

$$\begin{aligned}\frac{\text{term frequency}}{\text{document frequency}} &= \text{term frequency} \cdot \frac{1}{\text{document frequency}} \\ &= \text{term frequency} \cdot \text{inverse document frequency} \\ &= \text{tf} \cdot \text{idf}\end{aligned}$$

Now, there are more ways used to describe both term frequency and inverse document frequency. But the most common way is by putting them on a logarithmic scale:

$$tf(t,d) = log(1 + f_{t,d})$$

$$idf(t) = log(\frac{1+N}{1+n_t})$$

where:

$$\begin{aligned}f_{t,d} &= \text{count of term } \mathbf{t} \text{ in document } \mathbf{d} \\ N &= \text{total number of documents} \\ n_t &= \text{number of documents that contain term } \mathbf{t}\end{aligned}$$

We added 1 in the first logarithm to avoid getting $-\infty$ when $f_{t,d}$ is 0. In the second logarithm we added one fake document to avoid division by zero.

Before we transform our data into vectors of counts or tf-idf values we should remove English **stopwords**[6][7]. Stopwords are words that are very common in a language and are usually removed in the preprocessing stage of natural text-related tasks like sentiment analysis or search.

Note that we should construct our vocabulary only based on the training set. When we will process the test data in order to make predictions we should use only the vocabulary constructed in the training phase, the rest of the words will be ignored.

Now, let's create the data frames from the supplied csv files:

```
In [1]: import pandas as pd
```

In [2]:
```
# Read in the training and test datasets from previously created csv files

imdb_train = pd.read_csv('csv/imdb_train.csv')
imdb_test = pd.read_csv('csv/imdb_test.csv')
```

In [3]:
```
# Display information and first few entries from the training and test dataset
s

pd.set_option('display.max_colwidth', None)

print ("----- Training dataset Info:")
imdb_train.info(verbose=None, buf=None, max_cols=None, memory_usage=None, null
_counts=None)
print ("Training dataset Content:")
print(imdb_train.iloc[:5])

print ("\n----- Test dataset Info:")
imdb_test.info(verbose=None, buf=None, max_cols=None, memory_usage=None, null_
counts=None)
print ("Test dataset Content:")
print(imdb_test.iloc[:5])
```

```
----- Training dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   text    25000 non-null  object
 1   label   25000 non-null  int64
dtypes: int64(1), object(1)
memory usage: 293.0+ KB
Training dataset Content:

text  \
0
This is what I was expecting when star trek DS9 premiered. Not to slight DS9. That was a wonderful show in it's own right, however it never really gave the fans more of what they wanted. Enterprise is that show. While having a simila rity to the original trek it differs enough to be original in it's own ways. It makes the ideas of exploration exciting to us again. And that was one of t he primary ingredients that made the original so loved. Another ingredient to success was the relationships that evolved between the crew members. Viewers really cared deeply for the crew. Enterprise has much promise in this area as well. The chemistry between Bakula and Blalock seems very promising. While se xual tension in a show can often become a crutch, I feel the tensions on ente rprise can lead to much more and say alot more than is typical. I think when we deal with such grand scale characters of different races or species even, we get some very interesting ideas and television. Also, we should note the p erformances, Blalock is very convincing as Vulcan T'pol and Bacula really has a whimsy and strength of character that delivers a great performance. The res t of the cast delivered good performances also. My only gripes are as follow s. The theme. It's good it's different, but a little to light hearted for my liking. We need something a little more grand. Doesn't have to be orchestral. Maybe something with a little more electronic sound would suffice. And my one other complaint. They sell too many adds. They could fix this by selling less ads, or making all shows two parters. Otherwise we'll end up seeing the shows final act getting wrapped up way too quickly as was one of my complaints of V oyager.
1
This film reminded me so much of "A History of Violence" which pretended to b e a close study of violence and violent behavior but ended up just being noth ing short of a cheap action movie masquerading as some thinking film on viole nce. Dustin Hoffman and his new British bride move to a small English town an d encounter endless harassment from the local drunks who do nothing but hang at the pub all day and make trouble. Don't these men have a job? Anyway, Dust in takes all he can take and by the end of the film he holds up in his house and fights off each one of the drunk attackers by such gruesome means as boil ing whiskey poured over someone, feet being blown off by a shotgun and someon es head getting caught in a bear trap. Funny that someone would have a need f or such a large bear trap in a small British town except maybe put a mans hea d in it. Sam Peckinpah who made the "Wild Bunch" which also covered the topic of blood letting violence in which no one was spared. But it was done with st yle, and you believed it. Straw Dogs is not believable. First of all the loca tion is wrong and does not work. Why place it in England? I would think maybe in some inner city location or a small town in the American South in the 193 0's or something. Second it is not in my view ever really explained clearly w hy these men are so quick to violence except maybe they got drunk and felt a need to kill Hoffman and rape his wife. Sam Peckinpah missed the mark on this
```

one.
2
Theodore Rex is possibly the greatest cinematic experience of all time, and i s certainly a milestone in human culture, civilization and artistic expressio n!! In this compelling intellectual masterpiece, Jonathan R. Betuel aligns hi mself with the great film makers of the 20th century, such as Francis Ford Co pola, Martin Scorcese, Orson Welles and Roman Polanski. The special effects a re nothing less than breathtaking, and make any work by Spielberg look trite and elementary. At the time of it's release, Theodore Rex was such a revoluti onary gem that it raised the bar of film-making to levels never anticipated b y film makers. The concept of making not just a motion picture featuring a di nosaur, but adapting an action packed, thrilling detective novel, co-staring a "talking" dinosaur with a post-modern name such as "Theodore", and an exist ential female police officer changed humanity as we know it. The world could never be the same after experiencing such magnificent beauty. Watching Theodo re Rex is much akin to looking into the face of God and hearing Him say "you are my most beloved creation." This is one of the few films that is simply TO DIE FOR!!!
3
Although I didn't like Stanley & Iris tremendously as a film, I did admire th e acting. Jane Fonda and Robert De Niro are great in this movie. I haven't al ways been a fan of Fonda's work but here she is delicate and strong at the sa me time. De Niro has the ability to make every role he portrays into acting g old. He gives a great performance in this film and there is a great scene whe re he has to take his father to a home for elderly people because he can't ca re for him anymore that will break your heart. I wouldn't really recommend th is film as a great cinematic entertainment, but I will say you won't see much bette acting anywhere.
4 'De Grot' is a terrific Dutch thriller, based on the book written by Tim K rabbé. Another of his books, 'Het Gouden Ei' was made into the great Dutch my stery thriller called 'Spoorloos' ('The Vanishing') in 1988. This one is not as good as that thriller (although much better than the American remake also called 'The Vanishing') but there are times it comes close. Especially the op ening moments are terrific. We see a man, later we learn his name is Egon Wag ter (Fedja van Huêt), coming from a plane in Thailand. When he picks up his b ags it is pretty clear that he is smuggling something across the border. Thes e scenes are perfectly directed, photographed and acted. A kind of suspense i s created that you would normally not have in an opening scene like this. Lat er we see how Egon makes his deal in Thailand with a woman, both stating that they have never done anything like this. From this point the movie is constan tly flashback and flash-forward. We see how Egon, still as a child (here play ed by Erik van der Horst), befriends a guy named Axel (as a kid played by Ben ja Bruijning). We learn how they grew up as friends, sort of, and how Axel (a s an adult played by Marcel Hensema) became a criminal. Egon in the meanwhile goes to college and settles with a woman. Around this time he sometimes meets Axel but does not really want anything to do with him. The movie is chronolog ical in a way. It shows Egon and Axel as kids, than as students, young adult s, and in their mid-thirties. But from time to time, like I said, the movie g oes back to when they were kids and jumps forward again. Every time we see th em as kids it explains something that happens when they are adults. Minor spo ilers herein. The title means 'The Cave', and it is the cave that gives the m ovie its happy ending, although it is in fact not that happy. Like the beginn ing, the ending is terrific. The middle part of the movie is entertaining and in a way it distracts our attention of the first scenes, only to come back at that point in the end. It is the editing that gives the movie its happy endin g, although we can say the dramatic ending is happy in a way as well.

```
   label
0      1
1      0
2      1
3      1
4      1

----- Test dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   text    25000 non-null  object
 1   label   25000 non-null  int64
dtypes: int64(1), object(1)
memory usage: 293.0+ KB
Test dataset Content:

text  \
0
Progeny is about a husband and wife who experience time loss while making lov
e. Completely unaware of what this bizarre experience means they try to go on
with their lives. The hubby begins questioning the bizarre event and gets hel
p through a very annoying psychiatrist. He comes to believe that aliens are r
esponsible for this lapse in time and that the unborn baby he once thought wa
s his and his wife's actually belongs to the aliens. If ya ask me, this is a
great scifi/horror story. Taking a highly questionable real-life scenario inv
olving alien abduction and hybrid breeding is definite thumbs up from this gu
y. I love all things related to aliens and this story definitely delivered so
me good ideas. So if you also share an interest in things extraterrestrial, y
ou should be pretty happy with Progeny. At least story-wise anyways. Unfortun
ately the movie overall is pretty average. With average acting by all actors.
Yep, even by the consistently awesome Mr. Dourif, who still does deliver the
best performance. Though the black head doctor, delivers his lines really wel
l. There are a few points in the flick where some of the delivery is cringe o
r laugh worthy, which is fine in my book. I like them cheesy and this had a l
ittle bit of some nice stinky cheese, and I mean that in a good way. Anyways,
with a less than stellar script you can't really blame all the actors. I espe
cially didn't care for the Mother Hysteria the film went for. She wanted a ba
by so badly that she'd neglect and dismiss everything her loving husband (wh
o's a doctor!!) said to her. It almost reached a point where you actually did
n't care what happened to her. The Progeny is another flick by Brian Yuzna fr
om the icky-sticky film, Society. Again he delivers some slimy effects, and a
gain he delivers a pretty unique tale of horror. If you're into scifi/horror
or are a fan of Dourif and or Yuzna films, there's no real reason not to chec
k out this flick if you get the chance. A generous 7 outta 10.
1
This movie is very similar to Death Warrant with Jean-Claude Van Damme and al
so has some similarities to Island of Fire with Jackie Chan and I also heard
that there is some other very similar action movies, but this film has a much
better action than Death Warrant or even Island of Fire (that's right, the Ja
ckie Chan's movie). Rarely American action movies has such a great action seq
uences, though there was many negative reviews on this film, it easily beats
most of the action movies of that time who were more successful. There were m
any martial art's scenes, David Bradley was fast as Bruce Lee in this film an
d what else was good, that fighting scenes were much longer than in most of t
```

he American martial art's movies. The shoot-out scenes were similar with John Woo's movies, maybe not that good, but still very exiting. There was also man y impressive explosions and one great chase scene. I've seen some other David Bradley's movies, but this one, yet is the best in terms of action. OK, this movie has some cheesy moments, but which movie hasn't? The acting was decent, Charles Napier was incredible and his character was real tough. Adam Clark wh o played Squid and Yuji Okumoto who played the main bad guy were also very go od. Other actors acted pretty well too, though the acting isn't important in this type of movies. If you are action movies fan (I mean the real action mov ies fan, who really can appreciate the good action), than you must see this f ilm.
2
This is an excellent movie and I would recommend it to everyone. Mr. Drury's acting is top notch as it always is and he blends well with the other actors in the movie. Can't give away any of the suspense or drama found in the movi e. Hell to pay is a must see movie!!! The plot was very suspenseful. I would watch this movie over and over again because it has all the elements of a gre at western movie. It was very authentic in how they displayed the components dealing with this movie which includes the guns, horses, and clothing. The so undtrack is enjoyable and adds flavor to the movie. James Drury has the right touch when picking out a movie to be involved with. This is a another winner for the western genre. !!!!!
3 The 13th and last RKO Falcon film starts with the mutual injunction by Tom Conway as Tom Lawrence alias the Falcon and Ed Brophy as Goldie of "No dame s!" whilst they prepare to go on vacation. While you're still wondering what they're going on vacation from as they hadn't had a job since the beginning o f the 1st film in 1941 (with Sanders as Gay though and Jenkins as Goldie) the y bump into a woman and get dragged into a seedy industrial espionage caper. They promise to help her when her uncle is murdered, by taking an envelope co ntaining the details of a formula to make substitute industrial diamonds to h is business colleague in Miami. Suspect everyone here except the cops here wh o are after Lawrence and Goldie for the murder. To console himself Goldie k eeps paraphrasing travel brochures: "On the coldest day you can always enjoy the warmth of a nice cosy electric chair" for one. Some nice languid atmosphe ric nightclub scenes rub shoulders with some especially bad behaviour from th e baddies. Favourite bit: the dignified game of hide and seek/hunt the thimbl e the imperturbable and suave Lawrence has with the baddies on the sleeper tr ain. Least favourite bit: the most embarrassing scene in the entire series in the alligator wrestling hut definitely thrown in for the kids! All in all n ot the best in the series but yet another entertaining outing, with an overal l satisfying plot and many episodes even in this that make me wish they could have gone on for just a few more years as Columbia did with Boston Blackie, a lthough RKO were churning these out faster. Absolutely no sex, not much viole nce (in fact none at all by today's high standards), and positively no messag e all make this type of film anathema to serious people who can only regard m ovies as an art form that must depend on these three pillars. Three Diet Falc on's were made later with John Calvert in the title role, I don't mind them b ut could never bring myself to count them into the main series, which Tom Con way had made his own by this time. Sad also that it was all downhill after th is for Conway, who moved into TV, voice overs and even played Norman Conquest in Park Plaza 605 rather well in 1953. He also developed serious eye and alco hol problems I don't know if they were linked wound up poverty stricken a nd after a spell in hospital in 1967 was found dead in his girlfriend's bed. For us folk that want to at least we still have his 10 entertaining Falcon's plus a number of other worthy, even classic RKO movies from 1942 to 1946 with which to remember him by.
4

```
I LOVE Dr WHo SO much! I believe that David Tennant is the best Dr the show h
as ever had and Billie Piper the Best companion! I liked the way the Dr and R
ose had such a connection and a great relationship and the Dr came close a fe
w times to expressing his love for rose! It sadly came to an end after only 2
seasons. I will miss watching rose heaps and think that the show will not be
the same without Rose! But David is still there to make me laugh and make me
happy to watch him play this fantastic role! I rate this show 110% it is FANT
ASTIC! The graphics and monsters in this show are wonderful and every storyli
ne is different but somewhat connected and i have actually learned somethings
about love, the world and relationships from this show. Therefore it must be
one of the most fantastic shows of all time!
```

	label
0	1
1	1
2	1
3	1
4	1

Text vectorization

Fortunately, for the text vectorization part all the hard work is already done in the Scikit-Learn classes `CountVectorizer` [8] and `TfidfTransformer` [5]. We will use these classes to transform our csv files into unigram and bigram matrices(using both counts and tf-idf values). (It turns out that if we only use a n-gram for a large n we don't get a good accuracy, we usually use all n-grams up to some n. So, when we say here bigrams we actually refer to uni+bigrams and when we say unigrams it's just unigrams.) Each row in those matrices will represent a document (review) in our dataset, and each column will represent values associated with each word in the vocabulary (in the case of unigrams) or values associated with each combination of maximum 2 words in the vocabulary (bigrams).

`CountVectorizer` has a parameter `ngram_range` which expects a tuple of size 2 that controls what n-grams to include. After we constructed a `CountVectorizer` object we should call `.fit()` method with the actual text as a parameter, in order for it to learn the required statistics of our collection of documents. Then, by calling `.transform()` method with our collection of documents it returns the matrix for the n-gram range specified. As the class name suggests, this matrix will contain just the counts. To obtain the tf-idf values, the class `TfidfTransformer` should be used. It has the `.fit()` and `.transform()` methods that are used in a similar way with those of `CountVectorizer`, but they take as input the counts matrix obtained in the previous step and `.transform()` will return a matrix with tf-idf values. We should use `.fit()` only on training data and then store these objects. When we want to evaluate the test score or whenever we want to make a prediction we should use these objects to transform the data before feeding it into our classifier.

Note that the matrices generated for our train or test data will be huge, and if we store them as normal numpy arrays they will not even fit into RAM. But most of the entries in these matrices will be zero. So, these Scikit-Learn classes are using Scipy sparse matrices[9] (`csr_matrix` [10] to be more exactly), which store just the non-zero entries and save a LOT of space.

We will use a linear classifier with stochastic gradient descent, `sklearn.linear_model.SGDClassifier` [11], as our model. First we will generate and save our data in 4 forms: unigram and bigram matrix (with both counts and tf-idf values for each). Then we will train and evaluate our model for each these 4 data representations using `SGDClassifier` with the default parameters. After that, we choose the data representation which led to the best score and we will tune the hyper-parameters of our model with this data form using cross-validation in order to obtain the best results.

In [4]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
```

Unigram Counts

In [5]:
```
# Create a unigram vectorizer and process the training set to generate a list
 of words.
# Note that unigram processing is set via the ngram_range parameter

unigram_vectorizer = CountVectorizer(ngram_range=(1, 1))
unigram_vectorizer.fit(imdb_train['text'].values)
```

Out[5]:
```
CountVectorizer()
```

In [6]:
```
# Display the length and a few samples of the unigram vectorizer to show the w
ords that have been extracted

print("Number of words found:", len(unigram_vectorizer.get_feature_names()))
print(unigram_vectorizer.get_feature_names()[10000:10100])
print(unigram_vectorizer.get_feature_names()[50000:50100])
```

```
Number of words found: 74849
['bête', 'bûsu', 'bürgermeister', 'c1', 'c3', 'c3p0', 'c3po', 'c4', 'c57', 'c
__p', 'ca', 'caaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa', 'caallin
g', 'caan', 'caas', 'cab', 'cabal', 'caballe', 'caballeros', 'caballo', 'caba
ls', 'cabana', 'cabanne', 'cabaret', 'cabarnet', 'cabbage', 'cabbages', 'cabb
ie', 'cabby', 'cabel', 'cabell', 'cabells', 'cabin', 'cabinet', 'cabinets',
'cabins', 'cabiria', 'cable', 'cables', 'cabo', 'cabot', 'cabrón', 'cabs', 'c
aca', 'caccia', 'cache', 'cachet', 'cacho', 'cack', 'cackle', 'cackles', 'cac
kling', 'caco', 'cacophonist', 'cacophonous', 'cacophony', 'cacoyanis', 'caco
yannis', 'cacti', 'cactus', 'cactuses', 'cad', 'cada', 'cadaver', 'cadaverou
s', 'cadavers', 'cadavra', 'cadby', 'caddie', 'caddy', 'caddyshack', 'caden
a', 'cadence', 'cadences', 'cadet', 'cadets', 'cadfile', 'cadilac', 'cadilla
c', 'cadillacs', 'cadmus', 'cadre', 'cads', 'caduta', 'cady', 'caesar', 'caes
ars', 'caeser', 'caetano', 'cafe', 'cafes', 'cafeteria', 'caffari', 'caffeina
ted', 'caffeine', 'cafferty', 'caffey', 'café', 'cafés', 'cage']
['pincher', 'pinchers', 'pinches', 'pinching', 'pinchot', 'pinciotti', 'pin
e', 'pineal', 'pineapple', 'pineapples', 'pines', 'pinet', 'pinetrees', 'pine
yro', 'pinfall', 'pinfold', 'ping', 'pingo', 'pinhead', 'pinheads', 'pinho',
'pining', 'pinjar', 'pink', 'pinkerton', 'pinkett', 'pinkie', 'pinkins', 'pin
kish', 'pinko', 'pinks', 'pinku', 'pinkus', 'pinky', 'pinnacle', 'pinnacles',
'pinned', 'pinning', 'pinnings', 'pinnochio', 'pinnocioesque', 'pino', 'pinoc
chio', 'pinochet', 'pinochets', 'pinoy', 'pinpoint', 'pinpoints', 'pins', 'pi
nsent', 'pint', 'pinta', 'pinter', 'pintilie', 'pinto', 'pintos', 'pints', 'p
inup', 'pioneer', 'pioneered', 'pioneering', 'pioneers', 'piotr', 'pious', 'p
iovani', 'pip', 'pipe', 'piped', 'pipedream', 'pipeline', 'piper', 'pipers',
'pipes', 'piping', 'pippi', 'pippin', 'pipsqueak', 'piquant', 'piquantly', 'p
ique', 'piqued', 'piquer', 'piquor', 'piracy', 'pirahna', 'piranha', 'piranha
s', 'pirate', 'pirated', 'piraters', 'pirates', 'pirotess', 'pirouette', 'pir
ouettes', 'pirouetting', 'pirovitch', 'pirro', 'pis', 'pisa', 'pisana']
```

In [7]:
```
# Now process the training dataset to get a count of the words extracted earli
er

X_train_unigram = unigram_vectorizer.transform(imdb_train['text'].values)
```

In [8]:
```
# Display the attributes the word count matrix; notice it is huge with 25000 r
ows since we have 25000 entries
# in the training dataset and 74849 columns since we saw above that we have a
 vocabulary of 74849 words

print(repr(X_train_unigram))
```

```
<25000x74849 sparse matrix of type '<class 'numpy.int64'>'
        with 3431196 stored elements in Compressed Sparse Row format>
```

Unigram Tf-Idf

In [9]:
```
# Create a unigram tf-idf vectorizer and load the training set using the word
 count matrix from earlier

unigram_tf_idf_transformer = TfidfTransformer()
unigram_tf_idf_transformer.fit(X_train_unigram)
```

Out[9]:
```
TfidfTransformer()
```

In [10]:
```
# Now calculate the unigram tf-idf statistics

X_train_unigram_tf_idf = unigram_tf_idf_transformer.transform(X_train_unigram)
```

In [11]:
```
# Display the attributes the unigram tf-idf matrix; it should be the same size
as the unigram matrix above

print(repr(X_train_unigram_tf_idf))
```

```
<25000x74849 sparse matrix of type '<class 'numpy.float64'>'
        with 3431196 stored elements in Compressed Sparse Row format>
```

Bigram Counts

In [12]:
```
# Create a bigram vectorizer and process the training set to generate a list o
f bigrams.
# Note that bigram processing is set via the ngram_range parameter and so incl
udes unigrams and bigrams

bigram_vectorizer = CountVectorizer(ngram_range=(1, 2))
bigram_vectorizer.fit(imdb_train['text'].values)
```

Out[12]:
```
CountVectorizer(ngram_range=(1, 2))
```

In [13]:
```
# Display the length and a few samples of the bigram vectorizer to show the bi
grams that have been extracted

print("Number of bigrams found:", len(bigram_vectorizer.get_feature_names()))
print(bigram_vectorizer.get_feature_names()[10000:10100])
print(bigram_vectorizer.get_feature_names()[50000:50100])
```

```
Number of bigrams found: 1520266
['3am but', '3am invesment', '3am it', '3am taped', '3bs', '3bs who', '3d',
'3d adventure', '3d although', '3d and', '3d animated', '3d animation', '3d a
nimations', '3d animators', '3d artists', '3d assante', '3d bird', '3d bore',
'3d capabilities', '3d cg', '3d cgi', '3d character', '3d companies', '3d com
puter', '3d dept', '3d disgrace', '3d effect', '3d effects', '3d element', '3
d ending', '3d environment', '3d especially', '3d feel', '3d game', '3d game
s', '3d glasses', '3d grafics', '3d graphics', '3d gravity', '3d had', '3d im
ax', '3d in', '3d it', '3d just', '3d mario', '3d models', '3d movie', '3d mo
vies', '3d panoramic', '3d plat', '3d probably', '3d programs', '3d sequenc
e', '3d shoot', '3d shooter', '3d shooters', '3d special', '3d splatter', '3d
technology', '3d that', '3d trust', '3d was', '3d wave', '3d well', '3d whic
h', '3d with', '3d world', '3d wow', '3dvd', '3dvd collection', '3k', '3k al
l', '3k attack', '3k but', '3k if', '3k it', '3k not', '3lbs', '3lbs is', '3
m', '3m to', '3mins', '3mins jack', '3p', '3p felt', '3p uses', '3p wasn', '3
p0', '3p0 r2', '3pm', '3pm and', '3po', '3po and', '3po frank', '3po mind',
'3rd', '3rd 4th', '3rd act', '3rd and', '3rd annual']
['allen radiation', 'allen randall', 'allen really', 'allen reprises', 'allen
reprising', 'allen retains', 'allen retired', 'allen rivkin', 'allen screen',
'allen scripts', 'allen second', 'allen seems', 'allen sees', 'allen set', 'a
llen she', 'allen should', 'allen showed', 'allen shows', 'allen since', 'all
en slowly', 'allen small', 'allen smithee', 'allen soule', 'allen starrer',
'allen stereotypical', 'allen still', 'allen straight', 'allen stuff', 'allen
stuttering', 'allen style', 'allen such', 'allen surprisingly', 'allen surrog
ate', 'allen take', 'allen technique', 'allen tells', 'allen the', 'allen the
y', 'allen think', 'allen this', 'allen ticks', 'allen to', 'allen treat', 'a
llen waking', 'allen wannabe', 'allen warning', 'allen was', 'allen we', 'all
en when', 'allen whining', 'allen who', 'allen will', 'allen williams', 'alle
n with', 'allen work', 'allen would', 'allen writes', 'allen writing', 'allen
yes', 'allen you', 'allen young', 'allende', 'allende and', 'allende author',
'allende book', 'allende characters', 'allende has', 'allende killed', 'allen
de magical', 'allende not', 'allende novel', 'allende overthrown', 'allende s
hould', 'allende was', 'allende who', 'allens', 'allens stuff', 'allergic',
'allergic are', 'allergic reaction', 'allergic reactions', 'allergic to', 'al
lergies', 'allergies kept', 'allergy', 'allergy senator', 'allergy to', 'alle
s', 'alles action', 'alleviate', 'alleviate her', 'alleviate pain', 'alleviat
e the', 'alleviate their', 'alleviate visual', 'alley', 'alley 1912', 'alley
about', 'alley acting', 'alley again']
```

In [14]:
```
# Now generate bigram statistics on the training set

X_train_bigram = bigram_vectorizer.transform(imdb_train['text'].values)
```

In [15]:
```
# Display the attributes the bigram count matrix; notice it is really huge wit
h 25000 rows since we have 25000 entries
# in the training dataset and 1520266 columns since we saw above that we have
 1520266 bigrams

print(repr(X_train_bigram))
```

```
<25000x1520266 sparse matrix of type '<class 'numpy.int64'>'
        with 8689547 stored elements in Compressed Sparse Row format>
```

Bigram Tf-Idf

In [16]:
```
# Create a bigram tf-idf vectorizer and load the training set using the bigram
count matrix from earlier

bigram_tf_idf_transformer = TfidfTransformer()
bigram_tf_idf_transformer.fit(X_train_bigram)
```

Out[16]:
```
TfidfTransformer()
```

In [17]:
```
# Now calculate the bigram tf-idf statistics

X_train_bigram_tf_idf = bigram_tf_idf_transformer.transform(X_train_bigram)
```

In [18]:
```
# Display the attributes the bigram tf-idf matrix; it should be the same size
 as the bigram matrix above

print(repr(X_train_bigram_tf_idf))
```

```
<25000x1520266 sparse matrix of type '<class 'numpy.float64'>'
        with 8689547 stored elements in Compressed Sparse Row format>
```

Try the four different data formats (unigram, bigram with and without tf_idf) on the training set and pick teh best

Now, for each data form we split it into train & validation sets, train a `SGDClassifier` and output the score.

In [19]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import train_test_split
from scipy.sparse import csr_matrix
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f
1_score
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from typing import Tuple
```

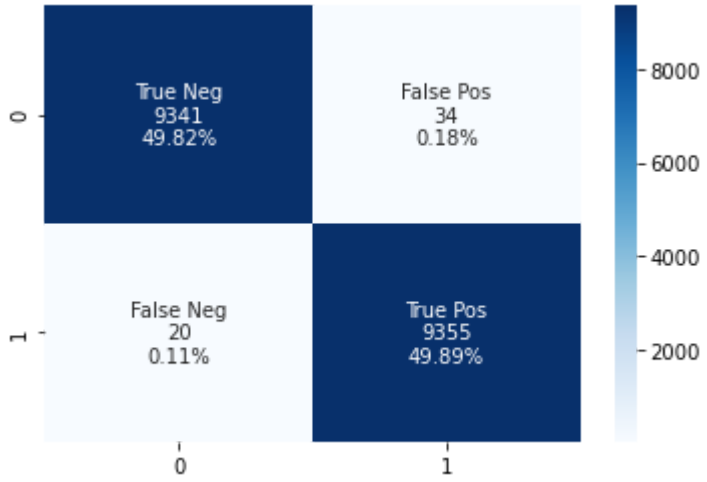
In [20]:
```
# Helper function to display confusion matrix

def display_confusion_matrix(y_true, y_pred) -> None:
    cf_matrix = confusion_matrix(y_true, y_pred)
    group_names = ['True Neg','False Pos','False Neg','True Pos']
    group_counts = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in cf_matrix.flatte
n()/np.sum(cf_matrix)]
    labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names,group_coun
ts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
    plt.show()
```

In [21]:
```
def train_and_show_scores(X: csr_matrix, y: np.array, title: str) -> Tuple[flo
at, float]:
    X_train, X_valid, y_train, y_valid = train_test_split(
        X, y, train_size=0.75, stratify=y
    )

    clf = SGDClassifier()
    clf.fit(X_train, y_train)
    train_score = clf.score(X_train, y_train)
    valid_score = clf.score(X_valid, y_valid)
    print(f'{title}\nTrain score: {round(train_score, 2)} ; Validation score:
{round(valid_score, 2)}')

    train_pred = clf.predict(X_train)
    valid_pred = clf.predict(X_valid)
    print(f'Train precision: {round(precision_score(y_train, train_pred), 2)}
 ; Validation precision: {round(precision_score(y_valid, valid_pred), 2)}')
    print(f'Train recall: {round(recall_score(y_train, train_pred), 2)} ; Vali
dation recall: {round(recall_score(y_valid, valid_pred), 2)}')
    print(f'Train F1: {round(f1_score(y_train, train_pred), 2)} ; Validation F
1: {round(f1_score(y_valid, valid_pred), 2)}')
    print("Train Confusion Matrix: ")
    print(confusion_matrix(y_train, train_pred))
    display_confusion_matrix(y_train, train_pred)
    print("Validation Confusion Matrix: ")
    print(confusion_matrix(y_valid, valid_pred))
    display_confusion_matrix(y_valid, valid_pred)
    print("\n")

    return train_score, valid_score, clf
```

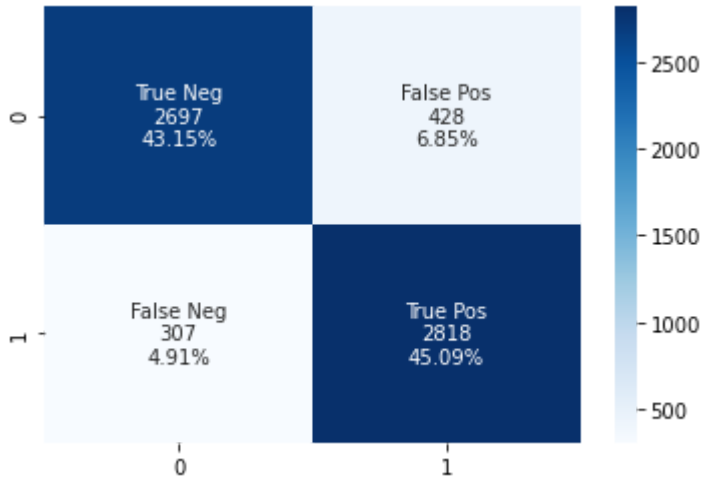
In [22]:
```
y_train = imdb_train['label'].values
```

In [23]:
```
uc_train_score, uc_valid_score, uc_clf = train_and_show_scores(X_train_unigram
, y_train, '----- Unigram Counts -----')
utfidf_train_score, utfidf_valid_score, utfidf_clf = train_and_show_scores(X_t
rain_unigram_tf_idf, y_train, '----- Unigram Tf-Idf -----')
bc_train_score, bc_valid_score, bc_clf = train_and_show_scores(X_train_bigram,
y_train, '----- Bigram Counts -----')
btfidf_train_score, btfidf_valid_score, btfidf_clf = train_and_show_scores(X_t
rain_bigram_tf_idf, y_train, '----- Bigram Tf-Idf -----')
```

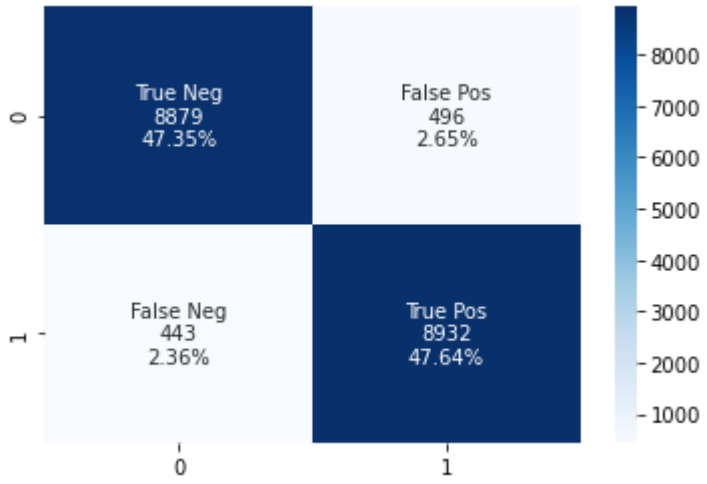
```
----- Unigram Counts -----
Train score: 1.0 ; Validation score: 0.88
Train precision: 1.0 ; Validation precision: 0.87
Train recall: 1.0 ; Validation recall: 0.9
Train F1: 1.0 ; Validation F1: 0.88
Train Confusion Matrix:
[[9341   34]
 [  20 9355]]
```



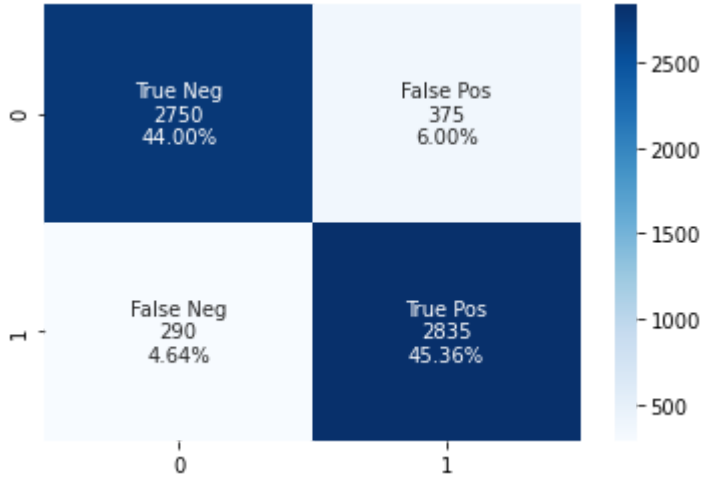

```
Validation Confusion Matrix:
[[2697  428]
 [ 307 2818]]
```



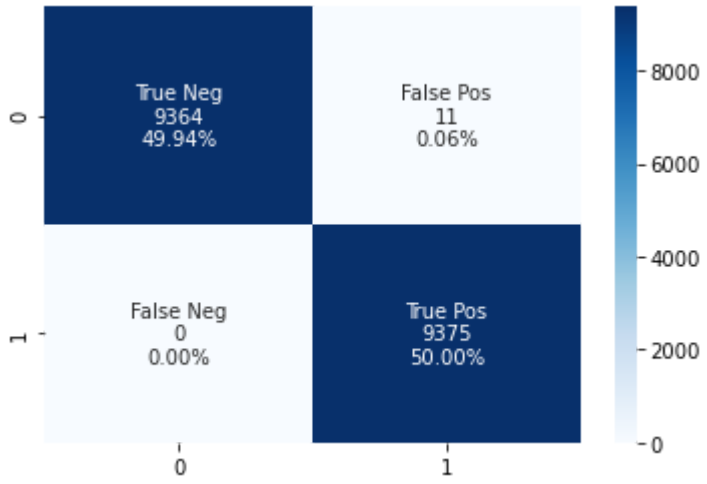

```
----- Unigram Tf-Idf -----
Train score: 0.95 ; Validation score: 0.89
Train precision: 0.95 ; Validation precision: 0.88
Train recall: 0.95 ; Validation recall: 0.91
Train F1: 0.95 ; Validation F1: 0.9
Train Confusion Matrix:
[[8879  496]
 [ 443 8932]]
```



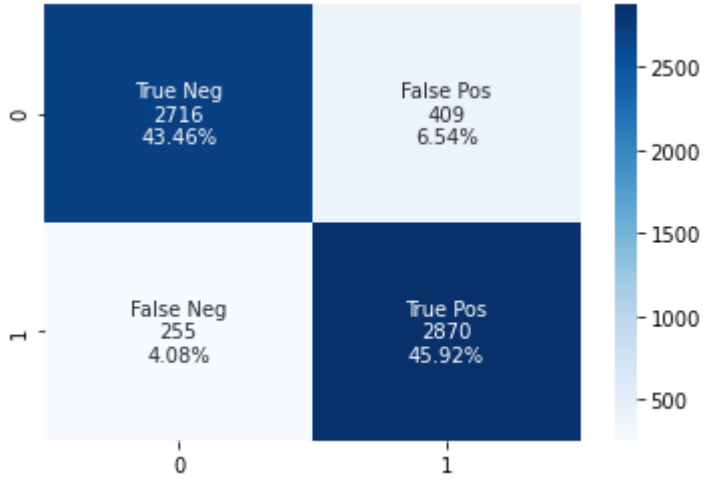
Validation Confusion Matrix:
[[2750 375]
 [290 2835]]



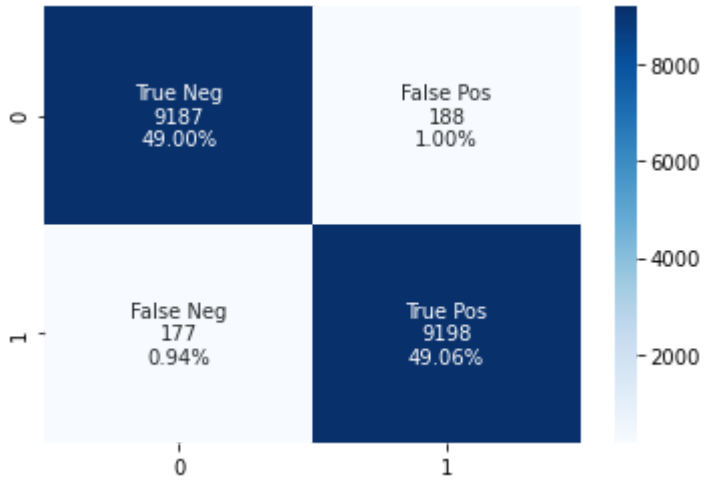
----- Bigram Counts -----
Train score: 1.0 ; Validation score: 0.89
Train precision: 1.0 ; Validation precision: 0.88
Train recall: 1.0 ; Validation recall: 0.92
Train F1: 1.0 ; Validation F1: 0.9
Train Confusion Matrix:
[[9364   11]
 [0 9375]]



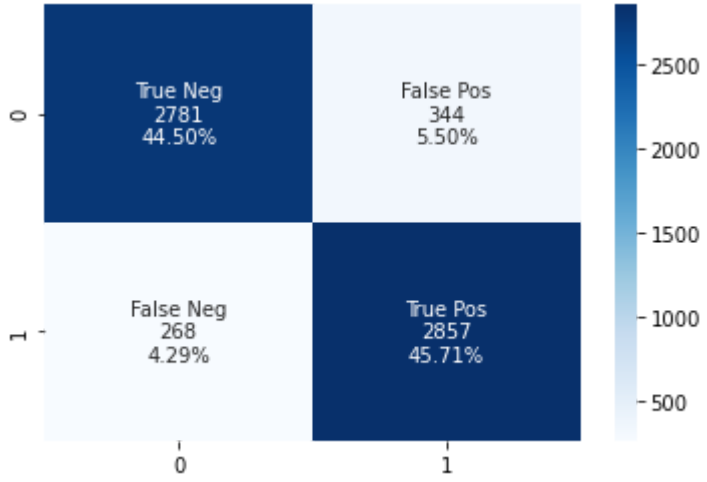

```
Validation Confusion Matrix:
[[2716  409]
 [ 255 2870]]
```




```
----- Bigram Tf-Idf -----
Train score: 0.98 ; Validation score: 0.9
Train precision: 0.98 ; Validation precision: 0.89
Train recall: 0.98 ; Validation recall: 0.91
Train F1: 0.98 ; Validation F1: 0.9
Train Confusion Matrix:
[[9187  188]
 [ 177 9198]]
```

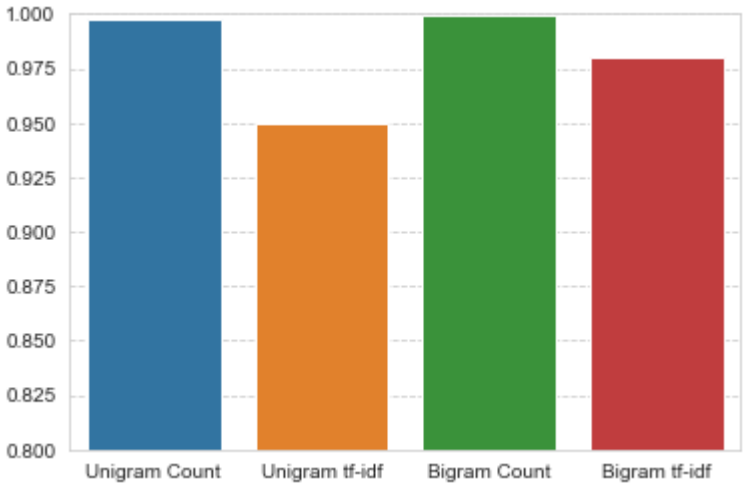



```
Validation Confusion Matrix:
[[2781  344]
 [ 268 2857]]
```

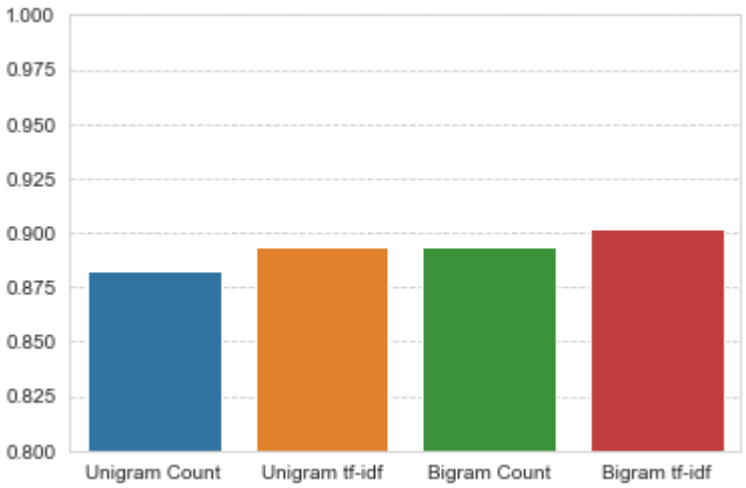


In [24]:
```
# Display the previously derived scores for the four scenarios

sns.set_style("whitegrid", {'grid.linestyle': '--'})

print ("Training score for the four approaches:")
ax1 = sns.barplot(
    x= ['Unigram Count', 'Unigram tf-idf', 'Bigram Count', 'Bigram tf-idf'],
    y= [uc_train_score, utfidf_train_score, bc_train_score, btfidf_train_score
])
ax1.set(ylim=(0.8, 1.0))
plt.show()

print ("Validation score for the four approaches:")
ax2 = sns.barplot(
    x= ['Unigram Count', 'Unigram tf-idf', 'Bigram Count', 'Bigram tf-idf'],
    y= [uc_valid_score, utfidf_valid_score, bc_valid_score, btfidf_valid_score
])
ax2.set(ylim=(0.8, 1.0))
plt.show()
```

Training score for the four approaches:



Validation score for the four approaches:



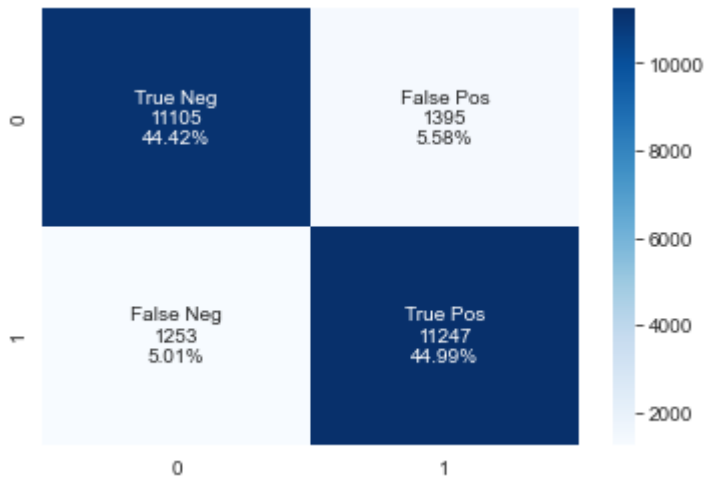
The best data form seems to be **bigram with tf-idf** as it gets the highest validation accuracy: **0.9**; so we will choose it as our preferred approach.

Testing model

In [25]:
```
# Transform the test data set into the bigram tf-idf format

X_test = bigram_vectorizer.transform(imdb_test['text'].values)
X_test = bigram_tf_idf_transformer.transform(X_test)
y_test = imdb_test['label'].values
```

In [26]:
```
# Now evaluate the test data using the previously trained bigram tf-idf classifier

clf = btfidf_clf
score = clf.score(X_test, y_test)

print(f'Score: {round(score, 4)}')

test_pred = clf.predict(X_test)
print(f'Test precision: {round(precision_score(y_test, test_pred), 4)}')
print(f'Test recall: {round(recall_score(y_test, test_pred), 4)}')
print(f'Test F1: {round(f1_score(y_test, test_pred), 4)}')
print("Test Confusion Matrix: ")
print(confusion_matrix(y_test, test_pred))
display_confusion_matrix(y_test, test_pred)
print("\n")
```

```
Score: 0.8941
Test precision: 0.8897
Test recall: 0.8998
Test F1: 0.8947
Test Confusion Matrix:
[[11105  1395]
 [ 1253 11247]]
```



And we got almost 90% test accuracy. That's not bad for our simple linear model. There are more advanced methods that give better results. The current state-of-the-art on this dataset is **97.42%** [13]

References

[1] Sentiment Analysis - Wikipedia (https://en.wikipedia.org/wiki/Sentiment_analysis)
[2] Learning Word Vectors for Sentiment Analysis (http://ai.stanford.edu/~amaas/papers/wvSent_acl2011.pdf)
[3] Bag-of-words model - Wikipedia (https://en.wikipedia.org/wiki/Bag-of-words_model)
[4] Tf-idf - Wikipedia (https://en.wikipedia.org/wiki/Tf%E2%80%93idf)
[5] TfidfTransformer - Scikit-learn documentation (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html)
[6] Stop words - Wikipedia (https://en.wikipedia.org/wiki/Stop_words)
[7] A list of English stopwords (https://gist.github.com/sebleier/554280)
[8] CountVectorizer - Scikit-learn documentation (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)
[9] Scipy sparse matrices (https://docs.scipy.org/doc/scipy/reference/sparse.html)
[10] Compressed Sparse Row matrix (https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html#scipy.sparse.csr_matrix)
[11] SGDClassifier - Scikit-learn documentation (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html)
[12] RandomizedSearchCV - Scikit-learn documentation (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html)
[13] Sentiment Classification using Document Embeddings trained with Cosine Similarity (https://www.aclweb.org/anthology/P19-2057.pdf)