Table of Contents:

Business Prop:

Most businesses in the world today have some kind of online presence with their customers. Many allow customers to leave a star rating and review of the product or service thay recieved. Most of their customers use these services as well with "91% of 18-34 year olds trust online reviews as much as personal recommendations, and 93% of consumers say that online reviews influenced

their purchase decisions." qualtrics (https://www.qualtrics.com/blog/online-review-stats/) This makes these reviews extremely valuable as far as data is concerned, it allows the producer to monitor what is and isn't working so they can adjust accordingly and it allows the customer to have a voice in a product or service they might like to keep using. However, due to the variability of human nature, the star rating system that often accompanies reviews is often not uniformly used. Such as reviews gettung an overly negative rating for mediocre service or an overly good rating for adequate service. The most ambiguous of these is the three star rating. Out of 1211 participants in on study said they feel a three star rating in a positive light. alittlehazebookblog (https://alittlehazebookblog.wordpress.com/2020/09/02/the-uncertainty-of-the-three-star-review-a-thought-piece/) While another stated that a "3.3 is the minimum star rating of a business consumers would engage with." qualtircs (https://www.qualtrics.com/blog/online-review-stats/) Which makes it seem like a much more negative rating. The aim of this project is to help companies utulize the data they already have, these reviews and rating systems, to disambiguate those three star reviews into data they can better use. To do this I will build a text based NLP binary classification model. This model will take in three star reviews and designate them as either positive or negative based on the actual text of the review.

In [1]:
```
import nltk
from nltk.classify.scikitlearn import SklearnClassifier
from nltk.classify import ClassifierI
nltk.download('punkt')
nltk.download('tagsets')
nltk.help.upenn_tagset()
from nltk import pos_tag
from nltk.stem import WordNetLemmatizer, PorterStemmer
from nltk.corpus import stopwords, wordnet
from nltk.tokenize import regexp_tokenize, word_tokenize, RegexpTokenizer
import random
import pickle #install
import pandas as pd
from sklearn.naive_bayes import MultinomialNB,BernoulliNB
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import StackingRegressor
from sklearn.metrics import accuracy_score, precision_score, confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, cross_val_score, cross_
from sklearn import metrics
from statistics import mode #install
import json
import csv
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from os import path
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from xgboost import XGBClassifier
from src.yelp import get_wordnet_pos, data_cleaner, num_to_cat, conf_matrix_p
```

```
    swiftly fiscally pitilessly ...
RBR: adverb, comparative
    further gloomier grander graver greater grimmer harder harsher
    healthier heavier higher however larger later leaner lengthier less-
    perfectly lesser lonelier longer louder lower more ...
RBS: adverb, superlative
    best biggest bluntest earliest farthest first furthest hardest
    heartiest highest largest least less most nearest second tightest wor
st
RP: particle
    aboard about across along apart around aside at away back before behi
nd
    by crop down ever fast for forth from go high i.e. in into just later
    low more off on open out over per pie raising start teeth that throug
h
    under unto up up-pp upon whole with you
SYM: symbol
    % & ' '' ''. ) ). * + ,. < = > @ A[fj] U.S U.S.S.R * ** ***
TO: "to" as preposition or infinitive marker
    to
```

Organizing the Data

The code below details how I read in and split 10,000 reivews for some inital EDA. The Yelp open source academic dataset contains about 8 million reviews. These reviews can be broken down into their user designated star ratings as follows: 3586460 five star reviews, 1673404 four star reviews, 842289 three star reviews, 635072 two star reviews and 1283897 one star reviews. To create a dataet that would fall into two classes I combined one and two stars reviews for my negative class and I combined four adn five star reivews for my positive class. As you can see from the numbers above this dataset skewed by having more positve reviews than negative at a 5:2 ratio. To reflect that distribution in my EDA datasets and to make this project mroe mangeable, I scaled down my data to 7500 positive reviews and 3000 negative reivews.

In [2]:
```
pos_7500 = open("C:/Users/edwardsrk/yelp_sentiment_analysis/pos_raw_list_2.tx
neg_3000 = open("C:/Users/edwardsrk/yelp_sentiment_analysis/neg_raw_list_2.tx
#read data in from stable text files
```

In [3]:
```
documents = []
pos_rev =[]
neg_rev = []

for r in pos_7500.split('\n'):
    documents.append( (r, "pos") )
    pos_rev.append( (r) )

for r in neg_3000.split('\n'):
    documents.append( (r, "neg") )
    neg_rev.append( (r) )

#split data on the newline and append to doc list with pos or neg tag
```

In [4]:
```
len(documents), len(neg_rev), len(pos_rev)
```

Out[4]: (10526, 3003, 7523)

In [5]:
```
all_words_raw = []

pos_7500_words = word_tokenize(pos_7500)
neg_3000_words = word_tokenize(neg_3000)
#tokenize data
```

In [6]:
```
for w in pos_7500_words:
    all_words_raw.append(w)
print(len(all_words_raw))

for w in neg_3000_words:
    all_words_raw.append(w)
print(len(all_words_raw))

#append tokenized data to all_words_raw
```

```
810870
1263624
```

EDA

In [7]:
```
print(len(all_words_raw))
#check length
```

```
1263624
```

In [8]:
```
tokens = all_words_raw
types = set(tokens)
len(types), len(tokens)

#create token and type vars
```

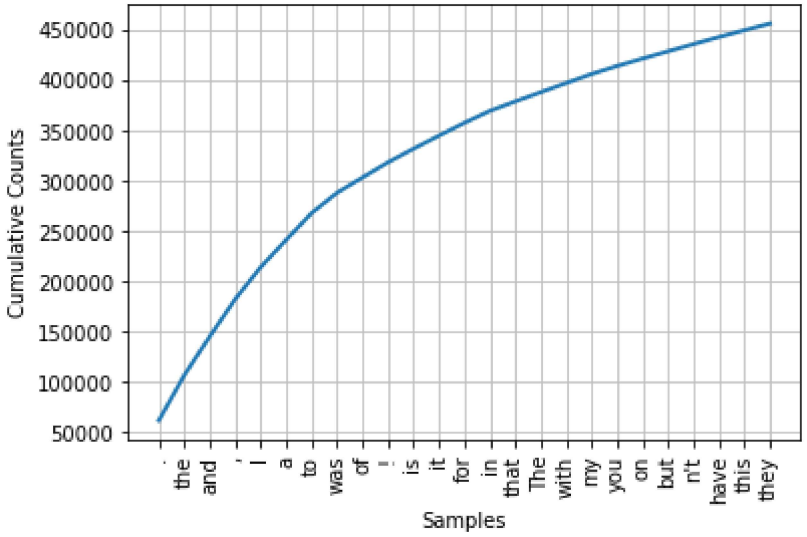
Out[8]:
```
(41906, 1263624)
```

In [9]:
```
fdist1 = nltk.FreqDist(tokens)
fdist1
#create frequency distribution, checks all tokens and how often they occur in
#prints out top 10 most used, these are usally stopwrods as the data has not
```

Out[9]:
```
FreqDist({'.': 62120, 'the': 45055, 'and': 38503, ',': 37129, 'I': 31782,
'a': 27019, 'to': 26697, 'was': 20187, 'of': 15183, '!': 14956, ...})
```

You can see from the frequencey distribution below that most of the most frequent words are stop words that will need to be stripped away later.

In [10]:
```
fdist1.plot(25, cumulative = True)
```



Out[10]: <AxesSubplot:xlabel='Samples', ylabel='Cumulative Counts'>

In [11]:
```
fdist1.hapaxes()[:10]
#first ten unique words
```

Out[11]:
```
['casa',
 'griping',
 'zips',
 'clotted',
 'ReviewI',
 'accommodating.Won',
 'Ton',
 'tons.Chicken',
 'Diced',
 'Cashews']
```

In [12]:
```
fdist1['the'], fdist1.freq('the')
#shows count of the word and frequency with which it appears
```

Out[12]: (45055, 0.03565538482966452)

The following cells show a few NLP techniques that show why cleaning your data is important and

that as you clean the amount of data you have goes down but that what is left is more lexically and semantically rich data. This more content heavy data is what the models will use to really do their predictions. Much of the other stuff that gets filtered out is considered to be noise.

In [13]:
```
words_lower = [w.lower() for w in all_words_raw]
#list comprehension for making all words lower case
```

In [14]:
```
len(words_lower)
```

Out[14]:
```
1263624
```

In [15]:
```
words_npunc = [w for w in words_lower if w.isalpha()]
len(words_npunc)
# list comp for all words that are alpha numeric, no punctuation
```

Out[15]:
```
1082114
```

In [16]:
```
fdist2 = nltk.FreqDist(words_npunc)
#check freq dist again
```

In [17]:
```
fdist2
#punctuation gone
```

Out[17]:
```
FreqDist({'the': 54329, 'and': 39332, 'i': 32763, 'a': 27758, 'to': 26898,
'was': 20336, 'it': 15939, 'of': 15268, 'is': 13634, 'for': 13177, ...})
```

In [18]:
```
#from nltk.corpus import stopwords
stop_words = stopwords.words('english')
#get a list of stop words from nltk
```

In the last sample cleaning technique below, stripping the stop words you can see we have pared downt he data from 1263624 words to 538646. That is almost cutting it in half. Doing htis allows the model to run faster while making more accurate predictions on real content words that distinguish the classes.

In [19]:
```
all_words_nstop = [w for w in words_npunc if not w in stop_words]
len(all_words_nstop)
#list comp to remove all stop words
```

Out[19]:
```
538646
```

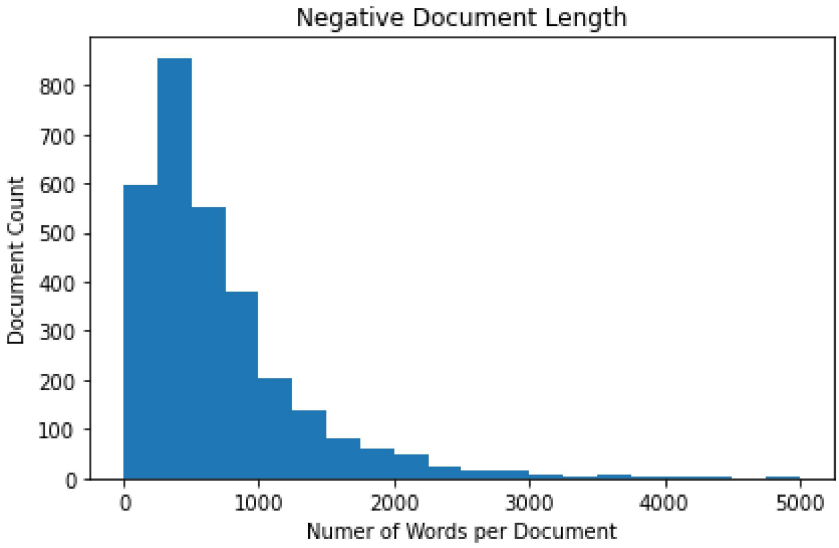
In [20]:
```
fdist3 = nltk.FreqDist(all_words_nstop)
fdist3
#thrid freqdist to check stopwords are gone
```

Out[20]:
```
FreqDist({'food': 5316, 'place': 5001, 'good': 4895, 'great': 4523, 'servic
e': 3788, 'time': 3417, 'would': 3412, 'like': 3349, 'one': 3315, 'get': 33
11, ...})
```

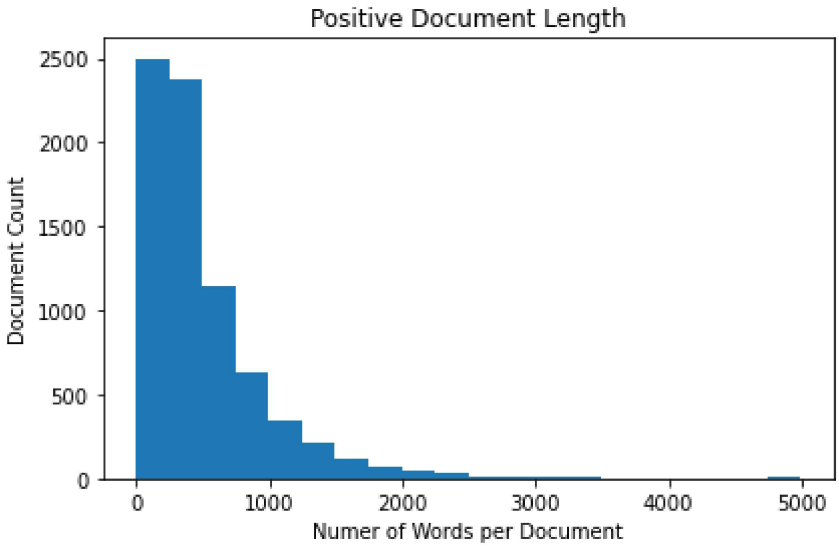
These two histograms show the distribution of words among each document within its class. As you can see, the positive reviews are much more vebose than the negative ones.

In [21]:
```
neg_doc_lengths = []
pos_doc_lengths = []
for doc in neg_rev:
    neg_doc_lengths.append(len(doc))

for doc in pos_rev:
    pos_doc_lengths.append(len(doc))
```

In [22]:
```
n_bins = 20
fig, axs = plt.subplots(1, sharey=True, tight_layout=True)
axs.hist(neg_doc_lengths, bins=n_bins)
axs.set_title("Negative Document Length");
axs.set_xlabel('Numer of Words per Document');
axs.set_ylabel("Document Count");
```



In [23]:
```
fig, axs = plt.subplots(1, sharey=True, tight_layout=True)
axs.hist(pos_doc_lengths, bins=n_bins)
axs.set_title("Positive Document Length");
axs.set_xlabel('Numer of Words per Document');
axs.set_ylabel("Document Count");
```



Data Cleaning

Data cleaning is required to make models run faster and have more accurate predicitons. In this notebook the cleaning steps I will use are: Part of speech tagging, casefolding, stripping stop words, stripping punctuation and lemmatizing.

In [24]:
```
df = pd.DataFrame(documents, columns =['text', 'tag'])
df.head()
#df.shape
#read data into df with text as col1 and tag as col2
```

Out[24]:

	text	tag
0	Oh happy day, finally have a Canes near my cas...	pos
1	I have been here twice. Very nice and laid bac...	pos
2	ORDER In (Delivery) ReviewI discovered this pl...	pos
3	A close friend was in town and so instead of t...	pos
4	Coconut's Fish Cafe is a fantastic, Five Stars...	pos

In [25]:
```
c_list = df.text.tolist()

clean_corpus = []
for doc in c_list:
    clean_corpus.append(data_cleaner(doc))
```

In [26]:
```
df['clean'] = clean_corpus
df = df.drop(['text'], axis= 1)
df.head()
```

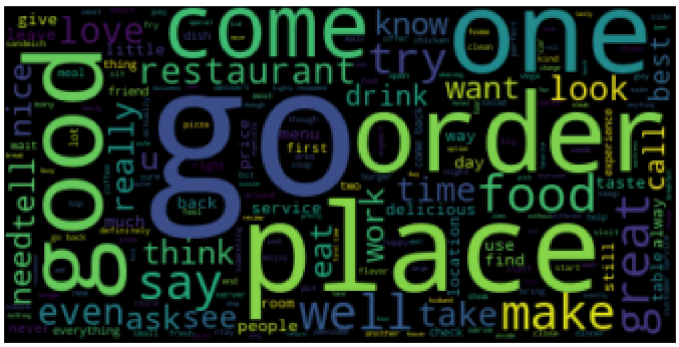
Out[26]:

	tag	clean
0	pos	oh happy day finally cane near casa yes others...
1	pos	twice nice lay back tried weekend southern men...
2	pos	order delivery reviewi discover place front de...
3	pos	close friend town instead take well establish ...
4	pos	coconut fish cafe fantastic five star fish caf...

Wordclouds

Wordclouds are a visualiztion similar to frequency distributions in that they showcase words that frequently appear. The bigger, bolder and more center a word is the higher frequency it has.

In [27]:
```
wordcloud_maker(df)
```



In [28]:
```
clean_pos = df.loc[df.tag == 'pos']
wordcloud_maker(clean_pos)
```

In [29]:
```
wordcloud_maker(clean_pos, stopwords = ['place', 'one', 'got', 'get'])
```

Good Features

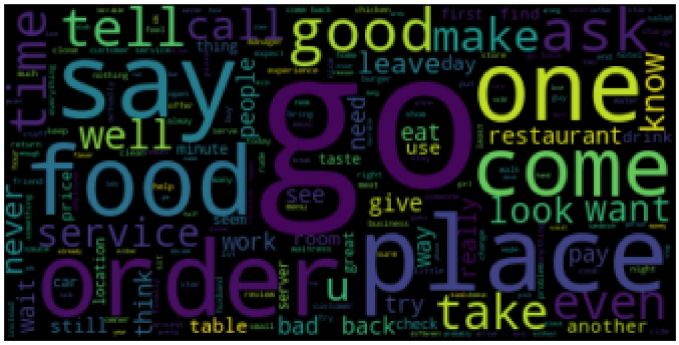
Below you can seee my third cleaned wordcloud. This word cloud takes ina list of stopwords created from its previous iterations, words that don't add much function or meaning to the idea of the class. Wors like 'got' or 'said' that don't really weigh in on whether the sentiemnt they are part of is negative or positive. Now that they have been stripped though you can see that the majority of the high frequency words are also high content and highly biased to one class or the other. Ideally these words here shown in the word cloud will also be the words or features the model uses to make its predictions.

In [30]:
```
wordcloud_maker(clean_pos, stopwords = ['place', 'one', 'got', 'get', 'said',
                                        'restaurant', 'people', 'would', 'time', 'a
                                        'make', 'said'])
```

Negative Wordcloud

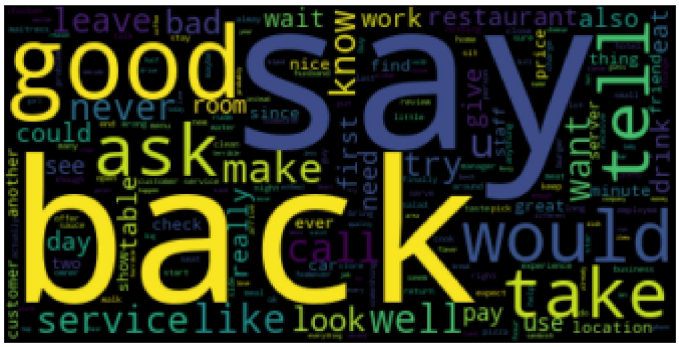
below are the wordclouds for my negative class. You may ntoice they look very similar to the positive wordclouds. This is because in the english langauge, we often express negation in two parts, the first being the negator followed byt what it is negating. For example, "that was not good" "I did not like that" "I wasn't impressed"

In [32]:
```
clean_neg = df.loc[df.tag == 'neg']
wordcloud_maker(clean_neg)
```



In [33]:
```
wordcloud_maker(clean_neg, stopwords = ['place', 'one', 'get', 'got' ])
```

In [34]:
```
wordcloud_maker(clean_neg, stopwords = ['place', 'one', 'get', 'got', 'next',
                                        'time', 'food', 'come', 'go', 'coming',
                                        'people', 'still', 'even', 'order', 'think'
                                        'much', 'way', 'may'])
```



Data for Modeling:

The data I used for modeling comes from a pickled random stratified sample. This data also comes from the Yelp academic data set and by virtueof the way it collected as a random stratifed sample, it maintians the 5:2 ratio for positive and negative data.

In [15]:
```
with open ('./data/pickles/40k_sample.p', "rb") as readfile:
    X, y = pickle.load(readfile)
```

In [16]:
```
df_strat = X.copy()
```

In [17]:
```
df_strat['stars'] = y
```

In [18]:
```
df_strat.head()
```

Out[18]:

	text	stars
7935619	Chips and salsa were awesome! Taco bowl was am...	4
6261538	When you take acrylic nails off, your suppose ...	1
1261782	One of the best places for Turkish food in the...	5
7802386	So difficult to book appointments with. They r...	2
6792587	Scheduled a "do what you gotta do" massage wit...	5

In [19]:
```
df_strat = df_strat[df_strat['stars'] != 3]
```

In [20]:
```
df_strat['tag'] = df_strat.stars.apply(num_to_cat)
```

In [21]:
```
c_list_strat = df_strat.text.tolist()

clean_corpus_strat = []
for docs in c_list_strat:
    clean_corpus_strat.append(data_cleaner(docs))
#Data gets cleaned of stopwords, punct, lemmatized POS tagged
```

In [22]:
```
df_strat['clean'] = clean_corpus_strat
df_strat = df_strat.drop(['text'], axis= 1)
df_strat.head()
```

Out[22]:

	stars	tag	clean
7935619	4	pos	chip salsa awesome taco bowl amaze big chunk c...
6261538	1	neg	take acrylic nail suppose soak cut away file n...
1261782	5	pos	one best place turkish food area interior bit ...
7802386	2	neg	difficult book appointment request come early ...
6792587	5	pos	schedule gotta massage kelsey great deal sore ...

TF-IDF

To get an idea of how well the data would perform with the data I do an inintial train test split and createa validation set just to see how the mdoels run and if more cleaning is needed. The cleaned

data will be passed to a TF-IDF vectorizer, I also tried a HashVectorizer for more speed and a CountVectorizer but niether performed as well.

In [23]:
```
data = df_strat['clean']
target = df_strat['tag']
#separate features from target
```

In [24]:
```
X_train, X_test, y_train, y_test = train_test_split(data,
                                                    target,
                                                    random_state=42,
                                                    test_size=0.25)
```

In [25]:
```
X_t, X_val, y_t, y_val = train_test_split(X_train, y_train,
                                          test_size=0.25, random_state=42)
# Secondary train-test split, validation set
```

In [13]:
```
vectorizer = TfidfVectorizer(analyzer = 'word', ngram_range=(1,2))
#calculate the TF-IDF score for unigrams and bigrams in text

X_t_vec = vectorizer.fit_transform(X_t)
X_t_vec  = pd.DataFrame.sparse.from_spmatrix(X_t_vec)
X_t_vec.columns = sorted(vectorizer.vocabulary_)
X_t_vec.set_index(y_t.index, inplace=True)
```

In [14]:
```
X_val_vec = vectorizer.transform(X_val)
X_val_vec  = pd.DataFrame.sparse.from_spmatrix(X_val_vec)
X_val_vec.columns = sorted(vectorizer.vocabulary_)
X_val_vec.set_index(y_val.index, inplace=True)
```

In [15]:
```
X_t_vec
```

Out[15]:

	aa	aa degree	aa look	aa quite	aa terminal	aaa	aaa appointment	aaa asks	aaa auto	aaa call	...	zwickle
7793664	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
4253932	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
7668463	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
6455785	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
5276025	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
6356400	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
5820686	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
7198942	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
2459980	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
4828491	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0

20190 rows × 583630 columns

In [16]:
```
vectorizer.vocabulary_
```

Out[16]:
```
{'favorite': 182538,
 'place': 380404,
 'vega': 549508,
 'away': 32860,
 'crazyness': 116536,
 'dub': 150972,
 'rest': 422074,
 'strip': 493336,
 'colorful': 98521,
 'parasol': 364263,
 'beautiful': 44236,
 'bar': 39895,
 'inside': 258103,
 'also': 11065,
 'equally': 163829,
 'easy': 153162,
 'eye': 176853,
 'outside': 358459,
 'patio': 368132,
```

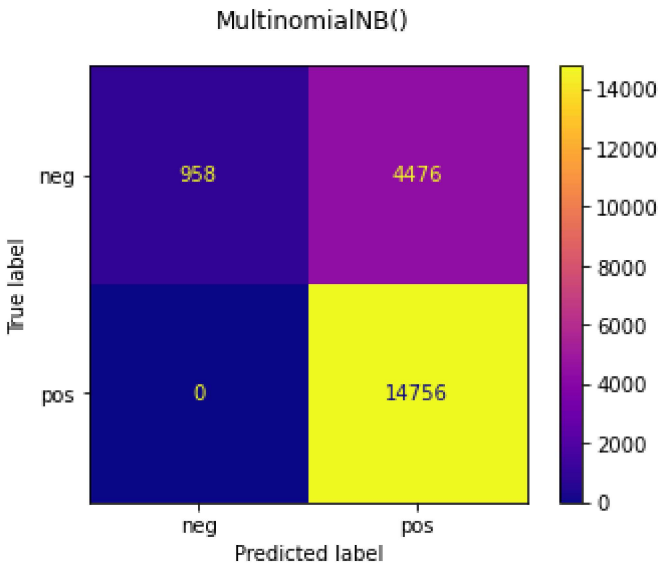
Modeling Data

In [17]:
```
%%time
# get accuracy of each model and compare, drop worst model/s for odd number o
# rank models for wieght
# use enseble method for to create THe MEGARITHM


mnb = MultinomialNB()
mnb.fit(X_t_vec, y_t)
mnb_pred = mnb.predict(X_val_vec)
mnb_acc = accuracy_score(y_val, mnb_pred)
mnb_acc
```

```
Wall time: 56.7 s
```

Out[17]:
```
0.7515601783060921
```

In [19]:
```
conf_matrix_plotter(mnb, X_t_vec, y_t)
```



In [20]:
```
%%time

bnb = BernoulliNB()
bnb.fit(X_t_vec, y_t)
bnb_pred = bnb.predict(X_val_vec)
#bnb.score(X_train, y_train), bnb.score(X_test, y_test)
bnb_acc = accuracy_score(y_val, bnb_pred)
bnb_acc

#conf_matrix_plotter(bnb, X_t_vec, y_t)
```

```
Wall time: 21 s
```

Out[20]:
```
0.7826151560178306
```

In [21]:
```
%%time

lr = LogisticRegression()
lr.fit(X_t_vec, y_t)
lr_pred = lr.predict(X_val_vec)
#lr.score(X_train, y_train), lr.score(X_test, y_test)
lr_acc = accuracy_score(y_val, lr_pred)
lr_acc
```

```
Wall time: 24 s
```

Out[21]:
```
0.924368499257058
```

FSM

First simple model, a vanilla SGD classifer that takes in clean, tf-idf vectorized data.

In [22]:
```
%%time

sgd = SGDClassifier()
sgd.fit(X_t_vec, y_t)
sgd_pred = sgd.predict(X_val_vec)
#sgd.score(X_train, y_train), sgd.score(X_test, y_test)
sgd_acc = accuracy_score(y_val, sgd_pred)
sgd_acc

#conf_matrix_plotter(sgd, X_t_vec, y_t)
```

```
Wall time: 21 s
```

Out[22]:
```
0.937295690936107
```

GridSearch

Finding the best model out of the eight listed below and tuning their hyperparameters.

In [10]:
```
lr = LogisticRegression()
mnb = MultinomialNB()
bnb = BernoulliNB()
knn =  KNeighborsClassifier()
dtc = DecisionTreeClassifier()
rfc = RandomForestClassifier()
xgb = XGBClassifier()
sgd = SGDClassifier()
```

In [24]:

```
##Logistic Regression Hyperparameters
lr_params = {
   'lr__C':[.1, .75],
   'lr__max_iter':[500],
   'lr__solver':['lbfgs', 'liblinear']

            }

##Multinomial Naive Bayes
mnb_params = {
    'mnb__alpha': [1, .5, .1],
    'mnb__fit_prior': ['True', 'False']
}

##Bernoulli Naive Bayes Hyperparameters
bnb_params = {'bnb__alpha': list(np.linspace(.1, 1, 10))}

##K Nearest Neighbors
knn_params = {
    'knn__n_neighbors': [3, 7, 12],
    'knn__weights':['uniform', 'distance'],
    'knn__p':[1, 2, 3, 4]
             }

##Decision Tree
dtc_params = {
    'dtc__criterion':['gini', 'entropy'],
    'dtc__max_depth': [10, 25, 40, 55],
    'dtc__min_samples_leaf':[1, 2, 3],
    'dtc__max_features': ['auto', 'sqrt']
              }

##Random Forest
rfc_params = {
    'rfc__n_estimators':[100, 125],
    'rfc__criterion':['gini', 'entropy'],
    'rfc__max_depth':[20, 40, None],
    'rfc__min_samples_leaf':[1, 2, 3],
    'rfc__max_features': ['auto','sqrt', 'log2']
                }

##XGBoost
xgb_params = {
        'xgb__eta':[.7, .9, 1.1],
        'xgb__max_depth': [6, 10, 15, 20],
        'xgb__learning_rate': [0.001, 0.01, 0.1, 0.2, 0,3],
        'xgb__subsample': [0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'xgb__colsample_bytree': [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'xgb__colsample_bylevel': [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'xgb__min_child_weight': [0.5, 1.0, 3.0, 5.0, 7.0, 10.0],
        'xgb__gamma': [0, 0.25, 0.5, 1.0],
        'xgb__reg_lambda': [0.1, 1.0, 5.0, 10.0, 50.0, 100.0],
        'xgb__n_estimators': [100, 120]
            }

##SGD Params
```

```
sgd_params ={
    'sgd__loss': ['hinge', 'log', 'modified_huber', 'perceptron', 'huber', 's
    'sgd__penalty': ['l2', 'l1', 'elasticnet'],
    'sgd__alpha': [0.0001, 0.001, 0.01, 0.1],
}
```

Instantiate Pipelines

In [11]:
```
lr_pipe = Pipeline(steps =[('tf', TfidfVectorizer()),('lr', LogisticRegressic
mnb_pipe = Pipeline(steps =[('tf', TfidfVectorizer()),('mnb', MultinomialNB()
bnb_pipe = Pipeline(steps =[('tf', TfidfVectorizer()),('bnb', BernoulliNB())]
knn_pipe = Pipeline(steps =[('tf', TfidfVectorizer()),('knn', KNeighborsClass
dtc_pipe = Pipeline(steps =[('tf', TfidfVectorizer()),('dtc', DecisionTreeCla
rfc_pipe = Pipeline(steps =[('tf', TfidfVectorizer()),('rfc', RandomForestCla
xgb_pipe = Pipeline(steps =[('tf', TfidfVectorizer()),('xgb', XGBClassifier()
sgd_pipe = Pipeline(steps =[('tf', TfidfVectorizer()),('sgd', SGDClassifier()
#creating  pipelines for each models
```

In [26]:
```
log_grid = GridSearchCV(estimator = lr_pipe, param_grid = lr_params, scoring
log_grid.fit(X_train, y_train)
log_grid.best_params_
```

Out[26]:
```
{'lr__C': 0.75, 'lr__max_iter': 500, 'lr__solver': 'liblinear'}
```

In [27]:
```
mnb_grid = GridSearchCV(estimator = mnb_pipe, param_grid = mnb_params, scorir
mnb_grid.fit(X_train, y_train)
mnb_grid.best_params_
```

Out[27]:
```
{'mnb__alpha': 0.1, 'mnb__fit_prior': 'True'}
```

In [28]:
```
bnb_grid = GridSearchCV(estimator = bnb_pipe, param_grid = bnb_params, scorir
bnb_grid.fit(X_train, y_train)
bnb_grid.best_params_
```

Out[28]:
```
{'bnb__alpha': 0.1}
```

In [29]:
```
knn_grid = GridSearchCV(estimator = knn_pipe, param_grid = knn_params, scorir
knn_grid.fit(X_train, y_train)
knn_grid.best_params_
```

```
C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\sklearn\mode
l_selection\_validation.py:610: FitFailedWarning: Estimator fit failed. T
he score on this train-test partition for these parameters will be set to
nan. Details:
Traceback (most recent call last):
  File "C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\skle
arn\model_selection\_validation.py", line 593, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\skle
arn\pipeline.py", line 346, in fit
    self._final_estimator.fit(Xt, y, **fit_params_last_step)
  File "C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\skle
arn\neighbors\_classification.py", line 179, in fit
    return self._fit(X, y)
  File "C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\skle
arn\neighbors\_base.py", line 466, in _fit
    raise ValueError("Metric '%s' not valid for sparse input. "
ValueError: Metric 'minkowski' not valid for sparse input. Use sorted(skl
earn.neighbors.VALID_METRICS_SPARSE['brute']) to get valid options. Metri
```

In [30]:
```
dtc_grid = GridSearchCV(estimator = dtc_pipe, param_grid = dtc_params, scorir
dtc_grid.fit(X_train, y_train)
dtc_grid.best_params_
```

Out[30]:
```
{'dtc__criterion': 'gini',
 'dtc__max_depth': 55,
 'dtc__max_features': 'auto',
 'dtc__min_samples_leaf': 3}
```

In [31]:
```
rfc_grid = GridSearchCV(estimator = rfc_pipe, param_grid = rfc_params, scorir
rfc_grid.fit(X_train, y_train)
rfc_grid.best_params_
```

Out[31]:
```
{'rfc__criterion': 'entropy',
 'rfc__max_depth': None,
 'rfc__max_features': 'auto',
 'rfc__min_samples_leaf': 1,
 'rfc__n_estimators': 100}
```

In [32]:
```
xgb_grid = GridSearchCV(estimator = xgb_pipe, param_grid = xgb_params, scorir
xgb_grid.fit(X_train, y_train)
xgb_grid.best_params_
```

In [33]:

```
sgd_grid = GridSearchCV(estimator = sgd_pipe, param_grid = sgd_params, scorir
sgd_grid.fit(X_train, y_train)
sgd_grid.best_params_
```

```
C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\sklearn\linear
_model\_stochastic_gradient.py:574: ConvergenceWarning: Maximum number of i
teration reached before convergence. Consider increasing max_iter to improv
e the fit.
  warnings.warn("Maximum number of iteration reached before "
C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\sklearn\linear
_model\_stochastic_gradient.py:574: ConvergenceWarning: Maximum number of i
teration reached before convergence. Consider increasing max_iter to improv
e the fit.
  warnings.warn("Maximum number of iteration reached before "
C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\sklearn\linear
_model\_stochastic_gradient.py:574: ConvergenceWarning: Maximum number of i
teration reached before convergence. Consider increasing max_iter to improv
e the fit.
  warnings.warn("Maximum number of iteration reached before "
C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\sklearn\linear
_model\_stochastic_gradient.py:574: ConvergenceWarning: Maximum number of i
teration reached before convergence. Consider increasing max_iter to improv
e the fit.
  warnings.warn("Maximum number of iteration reached before "
C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\sklearn\linear
_model\_stochastic_gradient.py:574: ConvergenceWarning: Maximum number of i
teration reached before convergence. Consider increasing max_iter to improv
e the fit.
  warnings.warn("Maximum number of iteration reached before "
```

Out[33]:

```
{'sgd__alpha': 0.0001, 'sgd__loss': 'modified_huber', 'sgd__penalty': 'l2'}
```

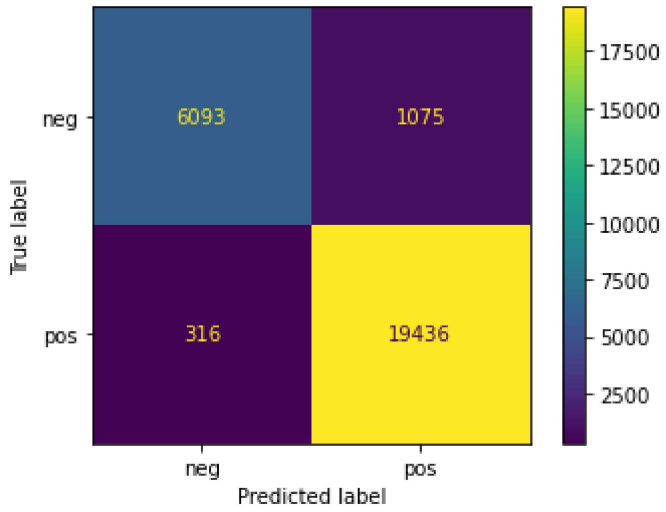
Tuned Models

In [34]:

```
lr_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer()),
                                 ('lr', LogisticRegression(C = 0.75, max_iter
#{'lr__C': 0.75, 'lr__max_iter': 500, 'lr__solver': 'liblinear'}


mnb_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer()),
                                  ('mnb', MultinomialNB(alpha = 0.1, fit_pric
#{'mnb__alpha': 0.1, 'mnb__fit_prior': 'True'}

bnb_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer()),
                                  ('bnb', BernoulliNB(alpha = 0.1))])
#{'bnb__alpha': 0.1}

knn_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer()),('knn', KNeighbor
#{'knn__n_neighbors': 12, 'knn__p': 2, 'knn__weights': 'uniform'}

dtc_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer()),
                                  ('dtc', DecisionTreeClassifier(criterion =
                                                                 min_samples_
#{'dtc__criterion': 'entropy','dtc__max_depth': 55,'dtc__max_features': 'auto


rfc_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer()),
                                  ('rfc', RandomForestClassifier(criterion =
                                                                 min_samples_l
#{'rfc__criterion': 'gini', 'rfc__max_depth': None, 'rfc__max_features': 'sqr
# 'rfc__min_samples_leaf': 1,'rfc__n_estimators': 125}

xgb_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer()),('xgb', XGBClassi

sgd_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer()),
                                  ('sgd', SGDClassifier(alpha = 0.0001, loss
#{'sgd__alpha': 0.0001, 'sgd__loss': 'modified_huber', 'sgd__penalty': 'elast
```

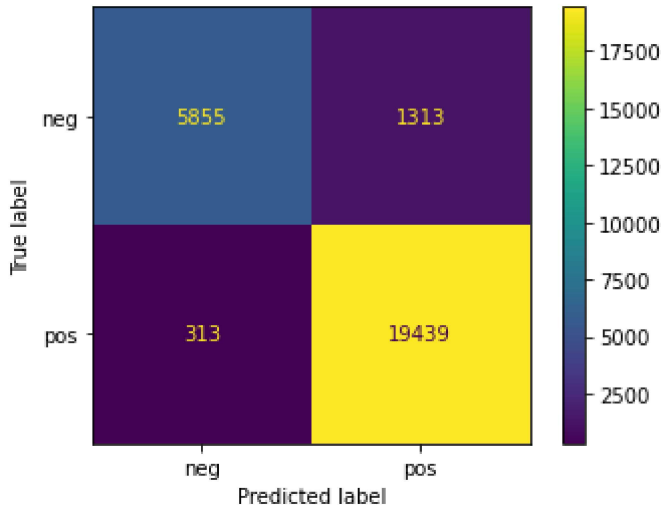
In [35]:
```
lr_pipe_tuned.fit(X_train, y_train);
plot_confusion_matrix(lr_pipe_tuned, X_train, y_train);
print("Mean Accuracy:", cross_val_score(lr_pipe_tuned, X_train, y_train).mean
```

```
Mean Accuracy: 0.9293090638930164
```



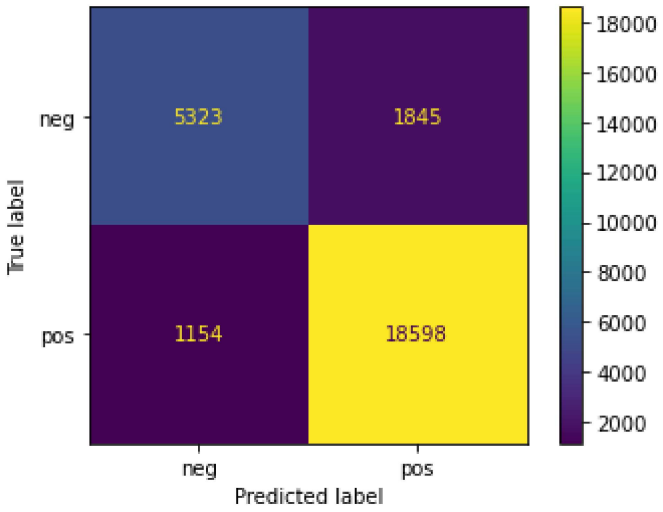
In [36]:
```
mnb_pipe_tuned.fit(X_train, y_train);
plot_confusion_matrix(mnb_pipe_tuned, X_train, y_train);
print("Mean Accuracy:", cross_val_score(mnb_pipe_tuned, X_train, y_train).mea
```

```
Mean Accuracy: 0.8947622585438335
```



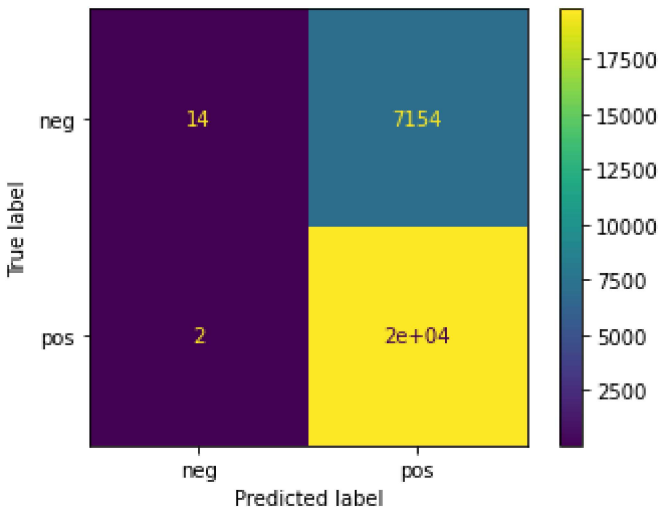
In [37]:
```
bnb_pipe_tuned.fit(X_train, y_train);
plot_confusion_matrix(bnb_pipe_tuned, X_train, y_train);
print("Mean Accuracy:", cross_val_score(bnb_pipe_tuned, X_train, y_train).mea
```

```
Mean Accuracy: 0.8521545319465081
```



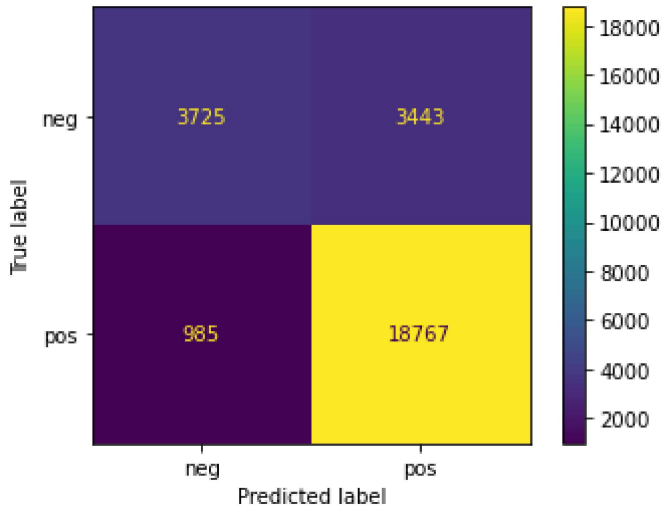
In [38]:
```
knn_pipe_tuned.fit(X_train, y_train);
plot_confusion_matrix(knn_pipe_tuned, X_train, y_train);
print("Mean Accuracy:", cross_val_score(knn_pipe_tuned, X_train, y_train).mea
```

```
Mean Accuracy: 0.7337295690936106
```



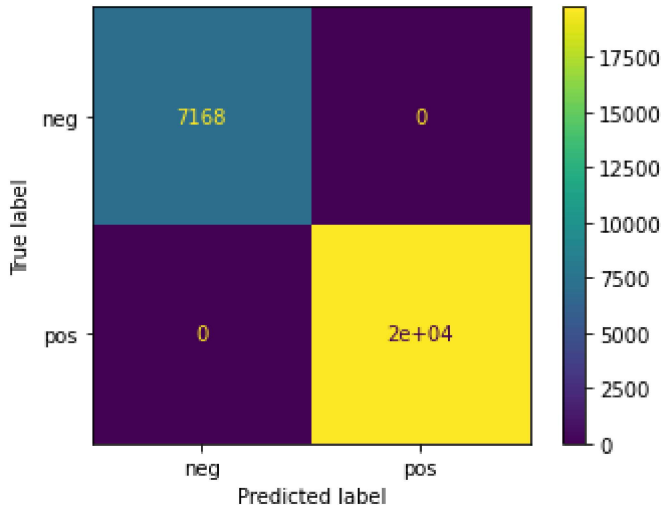
In [39]:
```
dtc_pipe_tuned.fit(X_train, y_train);
plot_confusion_matrix(dtc_pipe_tuned, X_train, y_train);
print("Mean Accuracy:", cross_val_score(dtc_pipe_tuned, X_train, y_train).mea
```

```
Mean Accuracy: 0.7790861812778603
```



In [40]:
```
rfc_pipe_tuned.fit(X_train, y_train);
plot_confusion_matrix(rfc_pipe_tuned, X_train, y_train);
print("Mean Accuracy:", cross_val_score(rfc_pipe_tuned, X_train, y_train).mea
```

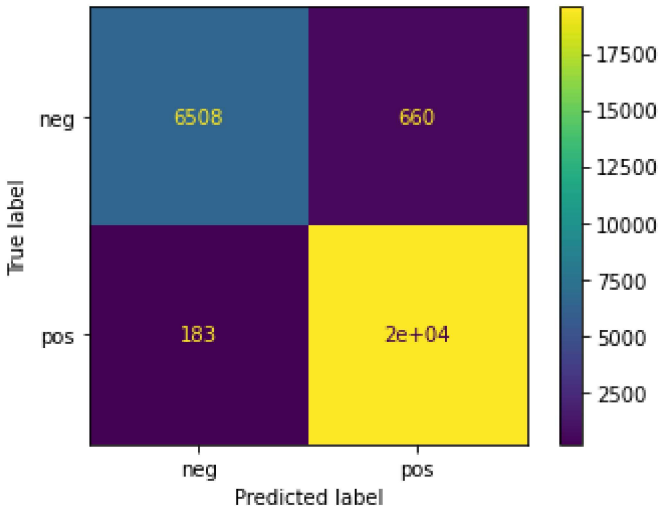
```
Mean Accuracy: 0.887072808320951
```



In [41]:
```
xgb_pipe_tuned.fit(X_train, y_train);
plot_confusion_matrix(xgb_pipe_tuned, X_train, y_train);
print("Mean Accuracy:", cross_val_score(xgb_pipe_tuned, X_train, y_train).mea
```

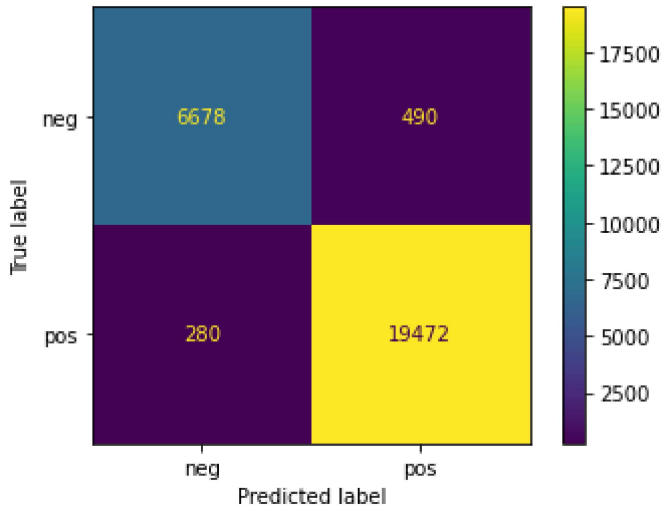
```
C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\sklear
n.py:888: UserWarning: The use of label encoder in XGBClassifier is depreca
ted and will be removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing XGBClas
sifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[12:15:55] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\sklear
n.py:888: UserWarning: The use of label encoder in XGBClassifier is depreca
ted and will be removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing XGBClas
sifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[12:16:07] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\sklear
n.py:888: UserWarning: The use of label encoder in XGBClassifier is depreca
ted and will be removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing XGBClas
sifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[12:16:17] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\sklear
n.py:888: UserWarning: The use of label encoder in XGBClassifier is depreca
ted and will be removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing XGBClas
sifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[12:16:27] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\skle
arn.py:888: UserWarning: The use of label encoder in XGBClassifier is dep
```

```
recated and will be removed in a future release. To remove this warning,
do the following: 1) Pass option use_label_encoder=False when constructin
g XGBClassifier object; and 2) Encode your labels (y) as integers startin
g with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[12:16:36] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\sklear
n.py:888: UserWarning: The use of label encoder in XGBClassifier is depreca
ted and will be removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing XGBClas
sifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[12:16:46] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
Mean Accuracy: 0.9190936106983655
```



In [42]:
```
sgd_pipe_tuned.fit(X_train, y_train);
plot_confusion_matrix(sgd_pipe_tuned, X_train, y_train);
print("Mean Accuracy:", cross_val_score(sgd_pipe_tuned, X_train, y_train).mea
```

```
Mean Accuracy: 0.9385215453194651
```



Vectorizer Tuning

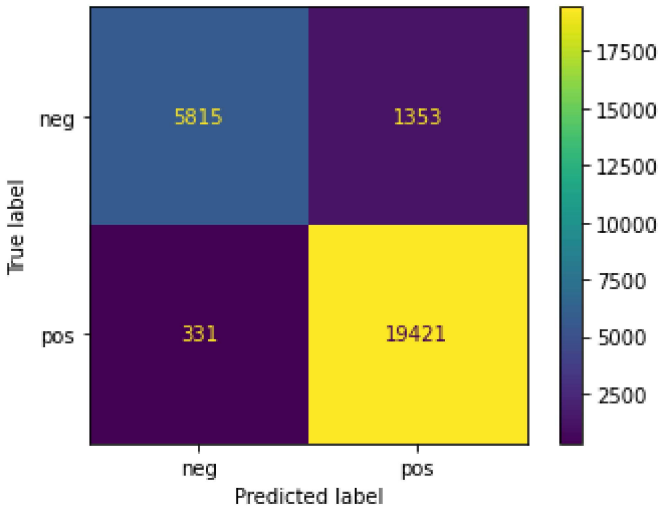
From the above models the three that did the best were tunes SGD,, tuned logistic regression and untuned XGB. These willbe the three models that will have hteir vectorizers tuned below.

In [68]:
```
lr_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer(max_features = 5000))
                                 ('lr', LogisticRegression(C = 0.75, max_iter
#Mean Accuracy: 0.9293090638930164

xgb_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer(max_features = 5000)
#Mean Accuracy: 0.9190936106983655

sgd_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer(max_features = 5000)
                                  ('sgd', SGDClassifier(alpha = 0.001, loss =
#Mean Accuracy: 0.9383358098068351
```

The first type of tuning I'm going to is try to minimize the number of important features needed to maintain the high accuracy score from above. Each model is tested with a lower and lower amount of featutres and seem to plateau until around 5000 features.

In [69]:
```
sgd_pipe_tuned.fit(X_train, y_train);
plot_confusion_matrix(sgd_pipe_tuned, X_train, y_train);
print("Mean Accuracy:", cross_val_score(sgd_pipe_tuned, X_train, y_train).mea
```

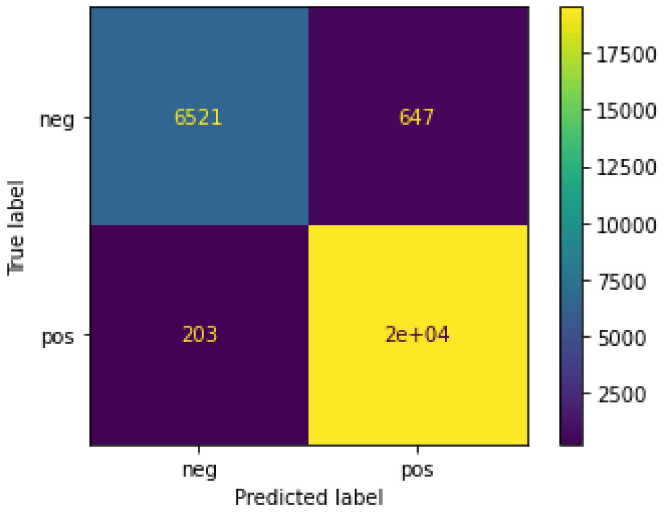
```
Mean Accuracy: 0.9277860326894503
```



In [70]:
```
xgb_pipe_tuned.fit(X_train, y_train);
plot_confusion_matrix(xgb_pipe_tuned, X_train, y_train);
print("Mean Accuracy:", cross_val_score(xgb_pipe_tuned, X_train, y_train).mea
```

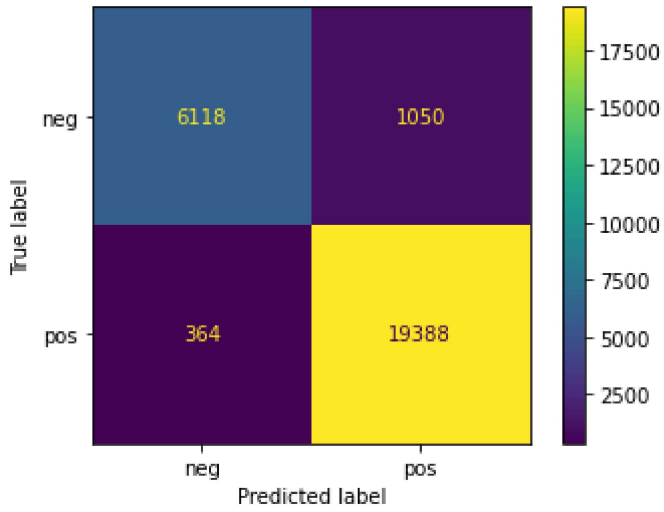
```
C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\sklear
n.py:888: UserWarning: The use of label encoder in XGBClassifier is depreca
ted and will be removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing XGBClas
sifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[15:24:17] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\sklear
n.py:888: UserWarning: The use of label encoder in XGBClassifier is depreca
ted and will be removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing XGBClas
sifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[15:24:25] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\sklear
n.py:888: UserWarning: The use of label encoder in XGBClassifier is depreca
ted and will be removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing XGBClas
sifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[15:24:31] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\sklear
n.py:888: UserWarning: The use of label encoder in XGBClassifier is depreca
ted and will be removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing XGBClas
sifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[15:24:38] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\skle
arn.py:888: UserWarning: The use of label encoder in XGBClassifier is dep
```

```
recated and will be removed in a future release. To remove this warning,
do the following: 1) Pass option use_label_encoder=False when constructin
g XGBClassifier object; and 2) Encode your labels (y) as integers startin
g with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[15:24:46] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\sklear
n.py:888: UserWarning: The use of label encoder in XGBClassifier is depreca
ted and will be removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing XGBClas
sifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[15:24:53] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
Mean Accuracy: 0.9189821693907876
```



In [71]:
```
lr_pipe_tuned.fit(X_train, y_train);
plot_confusion_matrix(lr_pipe_tuned, X_train, y_train);
print("Mean Accuracy:", cross_val_score(lr_pipe_tuned, X_train, y_train).mean
```

```
Mean Accuracy: 0.9316121842496285
```



Vectorizer Tuning: By Model

Logistic Regression

In [48]:
```
lr_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer(analyzer = 'word', ma
                                 ('lr', LogisticRegression(C = 0.75, max_iter
```

In [49]:
```
tf_params = {
    'tf__max_df': (0.25, 0.5, 0.75),
    'tf__ngram_range': [(1, 1), (1, 2), (1, 3)],
}
```

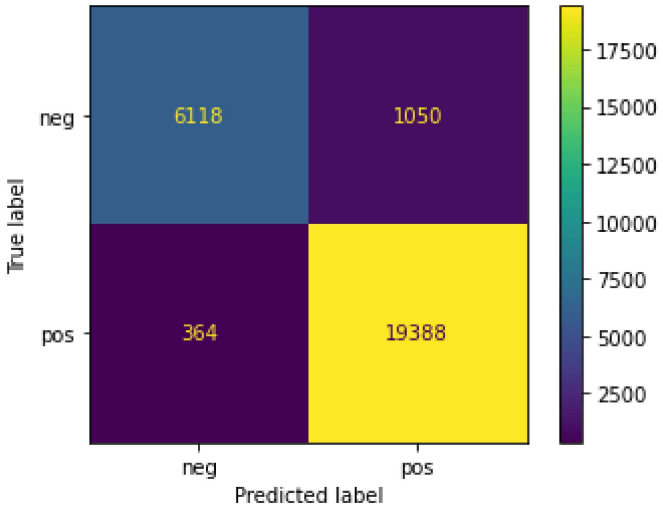
In [50]:
```
log_grid = GridSearchCV(estimator = lr_pipe_tuned, param_grid = tf_params, sc
log_grid.fit(X_train, y_train)
log_grid.best_params_
```

Out[50]:
```
{'tf__max_df': 0.5, 'tf__ngram_range': (1, 3)}
```

In [51]:
```
lr_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer(analyzer = 'word',max
                                 ('lr', LogisticRegression(C = 0.75, max_iter
```

In [72]:
```
lr_pipe_tuned.fit(X_train, y_train);
plot_confusion_matrix(lr_pipe_tuned, X_train, y_train);
print("Mean Accuracy:", cross_val_score(lr_pipe_tuned, X_train, y_train).mean
```

```
Mean Accuracy: 0.9316121842496285
```



XGB

In [53]:
```
xgb_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer(analyzer = 'word', m
```

In [54]:
```
xgb_grid = GridSearchCV(estimator = xgb_pipe_tuned, param_grid = tf_params, s
xgb_grid.fit(X_train, y_train)
xgb_grid.best_params_
```

```
C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\skle
arn.py:888: UserWarning: The use of label encoder in XGBClassifier is dep
recated and will be removed in a future release. To remove this warning,
do the following: 1) Pass option use_label_encoder=False when constructin
g XGBClassifier object; and 2) Encode your labels (y) as integers startin
g with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[13:31:13] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, th
e default evaluation metric used with the objective 'binary:logistic' was
changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd li
ke to restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\skle
arn.py:888: UserWarning: The use of label encoder in XGBClassifier is dep
recated and will be removed in a future release. To remove this warning,
do the following: 1) Pass option use_label_encoder=False when constructin
g XGBClassifier object; and 2) Encode your labels (y) as integers startin
g with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

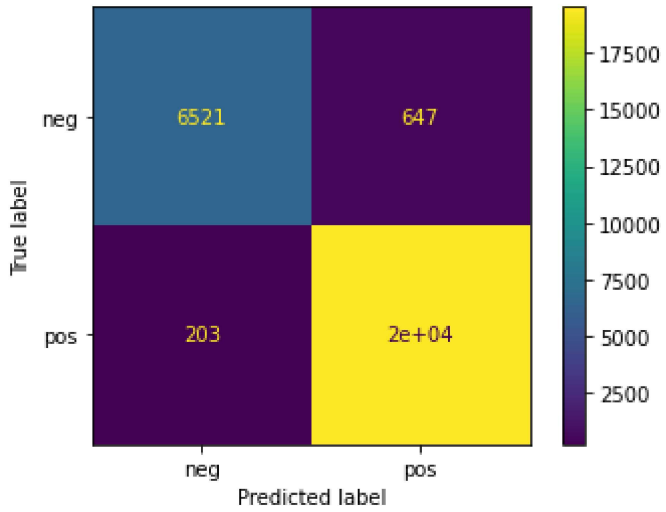
In [55]:

```
xgb_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer(analyzer = 'word',ma
                                                         ngram_range = (1,2),
```

In [73]:
```
xgb_pipe_tuned.fit(X_train, y_train);
plot_confusion_matrix(xgb_pipe_tuned, X_train, y_train);
print("Mean Accuracy:", cross_val_score(xgb_pipe_tuned, X_train, y_train).mea
```

```
C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\sklear
n.py:888: UserWarning: The use of label encoder in XGBClassifier is depreca
ted and will be removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing XGBClas
sifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[15:26:19] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\sklear
n.py:888: UserWarning: The use of label encoder in XGBClassifier is depreca
ted and will be removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing XGBClas
sifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[15:26:28] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\sklear
n.py:888: UserWarning: The use of label encoder in XGBClassifier is depreca
ted and will be removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing XGBClas
sifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[15:26:36] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\sklear
n.py:888: UserWarning: The use of label encoder in XGBClassifier is depreca
ted and will be removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing XGBClas
sifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[15:26:44] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\skle
arn.py:888: UserWarning: The use of label encoder in XGBClassifier is dep
```

```
recated and will be removed in a future release. To remove this warning,
do the following: 1) Pass option use_label_encoder=False when constructin
g XGBClassifier object; and 2) Encode your labels (y) as integers startin
g with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[15:26:52] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.

C:\Users\edwardsrk\anaconda3\envs\yelp_env\lib\site-packages\xgboost\sklear
n.py:888: UserWarning: The use of label encoder in XGBClassifier is depreca
ted and will be removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing XGBClas
sifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[15:27:01] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
Mean Accuracy: 0.9189821693907876
```



SGD

In [57]:
```
sgd_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer(max_features = 5000)
                                  ('sgd', SGDClassifier(alpha = 0.001, loss =
```

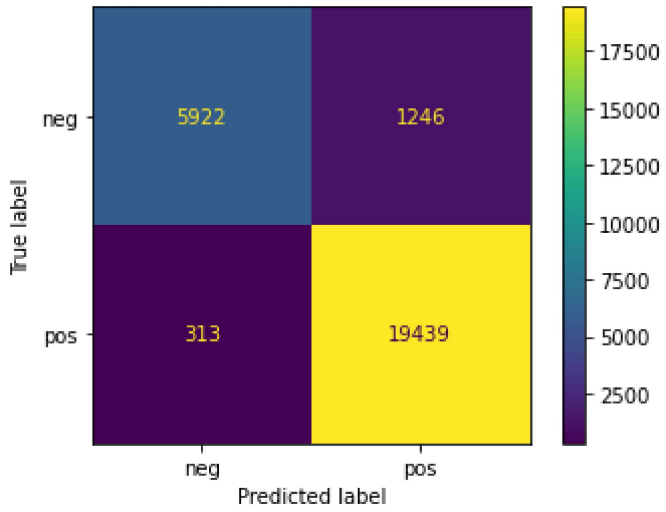
In [58]:
```
sgd_grid = GridSearchCV(estimator = sgd_pipe, param_grid = tf_params, scoring
sgd_grid.fit(X_train, y_train)
sgd_grid.best_params_
```

Out[58]:
```
{'tf__max_df': 0.5, 'tf__ngram_range': (1, 2)}
```

In [14]:
```
sgd_pipe_tuned = Pipeline(steps =[('tf', TfidfVectorizer(max_df = 0.75, ngram
                                  ('sgd', SGDClassifier(alpha = 0.001, loss =
```

In [15]:
```
sgd_pipe_tuned.fit(X_train, y_train);
plot_confusion_matrix(sgd_pipe_tuned, X_train, y_train);
print("Mean Accuracy:", cross_val_score(sgd_pipe_tuned, X_train, y_train).mea
```

```
Mean Accuracy: 0.9296805349182764
```

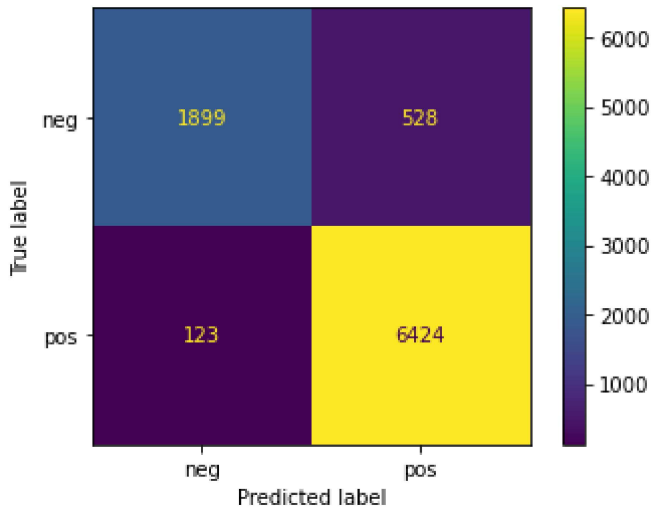


Final Model: Test Data

The final model I settled on was an SGD model and it performed at a 92% accuracy. It is a little lower than I anticiapated but still good overall and I think some of the that can be accounted for by the fact that you can have neutral statements in Enlgish. It still does a good job and hug the upper left corner of Roc graph nicely.

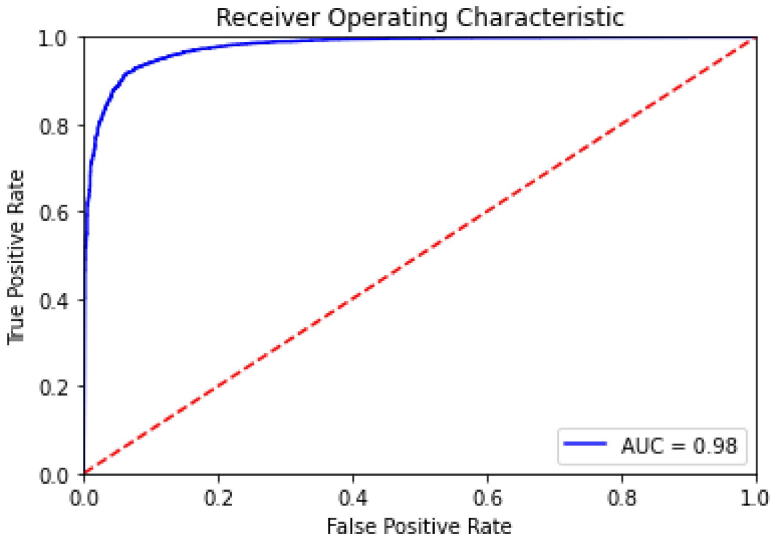
In [16]:
```
sgd_pipe_tuned.fit(X_train, y_train)

test_preds = sgd_pipe_tuned.predict(X_test)
sgd_acc = accuracy_score(y_test, test_preds)

plot_confusion_matrix(sgd_pipe_tuned, X_test, y_test);
print("Mean Accuracy:", cross_val_score(sgd_pipe_tuned, X_test, y_test).mean(
```

```
Mean Accuracy: 0.924560481704723
```



In [18]:
```
#tn,fp,
#fn, tp

fpr = 527/ (527+1900)
tpr = 6422/ (6422+125)

probs = sgd_pipe_tuned.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds, pos_label = 'pos')
roc_auc = metrics.auc(fpr, tpr)
```

In [19]:
```
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [20]:
```
features = sgd_pipe_tuned.steps[0][1].get_feature_names()
feat_values = sgd_pipe_tuned[1].coef_
```

In [21]:
```
d = {'features' : features}
feats = pd.DataFrame(data = d)
feats['values'] = feat_values[0]
```

In [22]:
```
sorted_feats = feats.sort_values(by='values')
```

Key Features

Below are the features associated with the highest and lowest coefficients. The lower the coefficient the higher influence those words are to the negative classand the higher the coefficient the higher influence those words have to the positive class. The features on either end of the coefficent spectrum make sense. The most interesting find is that the word 'pay' is so highly correlated with being negative as paying is a standard event associated with using a business or service.

In [23]:
```
sorted_feats.head()
```

Out[23]:

	features	values
298	bad	-1.708061
2140	horrible	-1.503607
4378	terrible	-1.489513
3699	rude	-1.458982
3140	pay	-1.283361

In [24]:
```
sorted_feats.tail()
```

Out[24]:

	features	values
124	amaze	1.526333
383	best	1.741395
1047	delicious	1.752888
2559	love	1.873046
1954	great	2.257790

Pickle Model

In [25]:
```
filename = ('final_sgd_model')
pickle.dump(sgd_pipe_tuned, open(filename, 'wb'))
#pickle model for later
```

In [26]:
```
#loaded_model = pickle.load(filename, 'rb'))
#result = loaded_model.predict(X_test, y_test)
#load pickle model
```

Grubhub Review Data

To test my model out on actual 3 star reviews I had to compile my own corpus of tagged data. In another notebook call INSERT NAME HERE, I used Selenium along with a custom scraper to scrape data from the popular food ordering app called Grubhub. The data I scraped includes a star rating, a text review and sentiment tag provided by Grubhub as positive, negative or neutral. This dataset allows me to test my model against tagged 3 star reviews as a method of evalutation. The reviews are all of local Seattle restaurants and the set contains about 2500 reviews rated from 1 to 5 stars.

In [36]:
```
gb_df = pd.read_csv("C:/Users/edwardsrk/final_proj/yelp_reviews_tensorflow/da
```

In [37]:
```
gb_df =gb_df.drop('Unnamed: 0', axis = 1)
gb_df.head()
```

Out[37]:

	stars	tags	text
0	3	Positive	Food was good but it came an hour early.
1	1	Negative	Meh. And driver was blasting music in his car
2	4	Positive	Best Thai food nearby!
3	4	Positive	Great food! Just not a timely delivery
4	4	Positive	The food was delicious.

In [38]:
```
one_star = gb_df[gb_df['stars'] == 1]
two_star = gb_df[gb_df['stars'] == 2]
three_star = gb_df[gb_df['stars'] == 3]
four_star = gb_df[gb_df['stars'] == 4]
five_star = gb_df[gb_df['stars'] == 5]
```

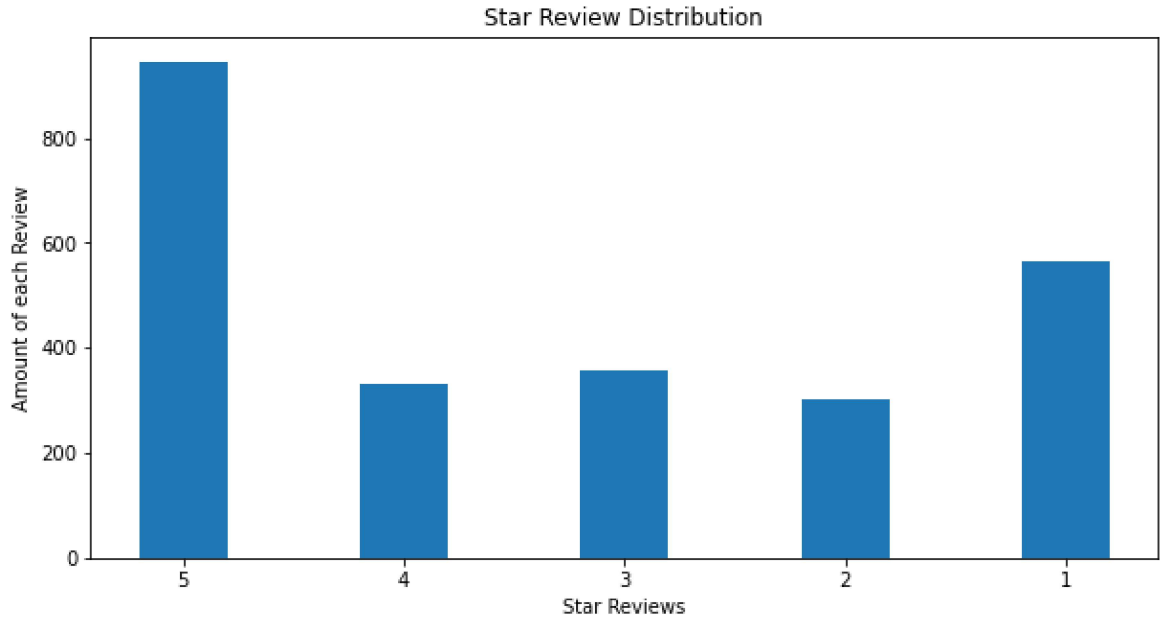
In [39]:
```
len(five_star), len(four_star), len(three_star), len(two_star), len(one_star)
```

Out[39]: (945, 331, 356, 304, 566)

Grubhub EDA

```
The ratio of reviews for the over all data set is 3 positive to 2 negative.
```

In [40]:
```
lens = {'5' :len(five_star), '4':len(four_star), '3' :len(three_star), '2' :l

x = list(lens.keys())
y = list(lens.values())

fig = plt.figure(figsize = (10, 5))

plt.bar(x,y, width = 0.4)

plt.xlabel("Star Reviews")
plt.ylabel("Amount of each Review")
plt.title("Star Review Distribution")
plt.show()
```

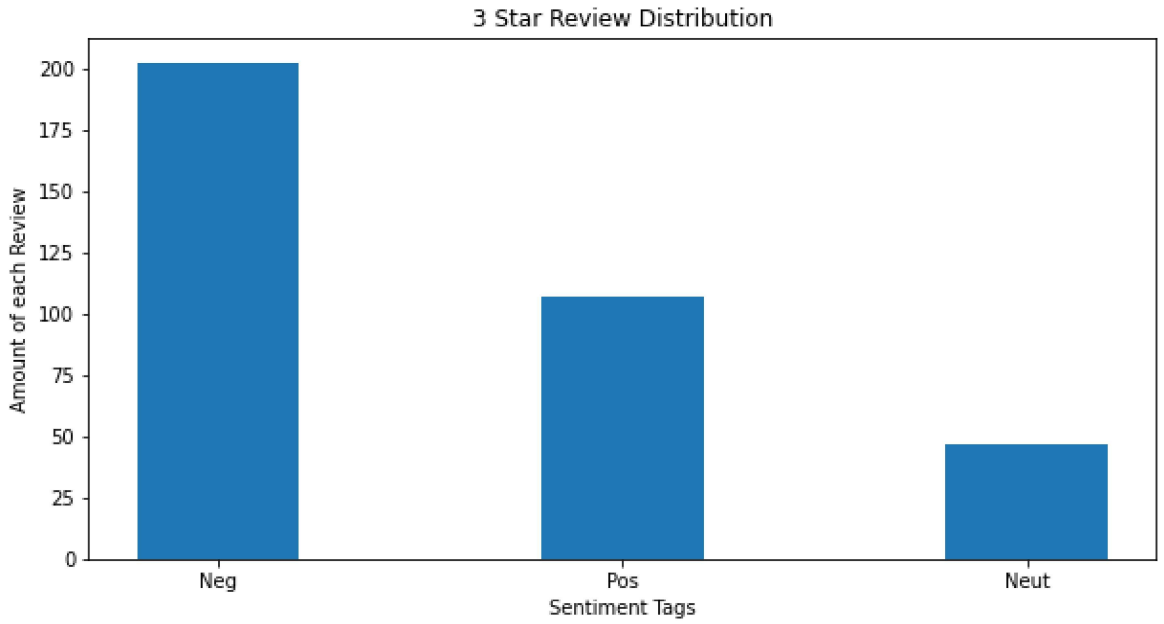


In [41]:
```
three_star.tags.value_counts()
```

Out[41]:
```
Negative    202
Positive    107
Neutral      47
Name: tags, dtype: int64
```

The ratio of negative reviews to positive is about 2 negative revies to one positive review. This wiill drastically and negatively impact how well the model does. The model was trained on a ratio or 5 positive to 2 negative.

In [42]:
```
dist_list = list(three_star.tags.value_counts())
data = {'Neg' : dist_list[0], 'Pos': dist_list[1], 'Neut': dist_list[2]}
x = list(data.keys())
y = list(data.values())

fig = plt.figure(figsize = (10, 5))

plt.bar(x,y, width = 0.4)

plt.xlabel("Sentiment Tags")
plt.ylabel("Amount of each Review")
plt.title("3 Star Review Distribution")
plt.show()
```



In [43]:
```
df_gb_split = [four_star, five_star, one_star, two_star]
df_gb = pd.concat(df_gb_split)
#concat all for extra test set
```

In [44]:
```
df_gb.stars.value_counts()
```

Out[44]:
```
5    945
1    566
4    331
2    304
Name: stars, dtype: int64
```

In [45]:
```
three_star_binary = three_star[three_star.tags != 'Neutral']
```

In [46]:
```
three_star_binary.tags.value_counts()
```

Out[46]:
```
Negative    202
Positive    107
Name: tags, dtype: int64
```

In [47]:
```
df_gb = df_gb[df_gb.tags != 'Neutral']
df_gb.tags.value_counts()
```

Out[47]:
```
Positive    1115
Negative     859
Name: tags, dtype: int64
```

In [48]:
```
grub = gb_cleaner(df_gb)
grub.head()
```

Out[48]:

	tag	clean
2	pos	best thai food nearby
3	pos	great food timely delivery
4	pos	food delicious
7	pos	excellent steam vegetable pharmacy see lew
8	pos	food great packaging much sauce soup spill poo...

In [49]:
```
df_gb_pos = grub[grub.tag != 'neg']
df_gb_neg = grub[grub.tag != 'pos']
```

In [50]:
```
wordcloud_maker(df_gb_pos)
```

The most prominent negative word shere differ greatly thanthose rpesented within the Yelp review dataset. These are much more focused on the performance of Grubhub and their drivers than on the business itself.

In [51]:
```
wordcloud_maker(df_gb_neg)
```

In [52]:
```
grub_3 = gb_cleaner(three_star_binary)
wordcloud_maker(gb_cleaner(three_star_binary))
```

```
C:\Users\edwardsrk\final_proj\yelp_reviews_tensorflow\src\yelp.py:111: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  df['tag'] = df.tags.apply(retagger)
C:\Users\edwardsrk\final_proj\yelp_reviews_tensorflow\src\yelp.py:119: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  df['clean'] = clean_corpus
```

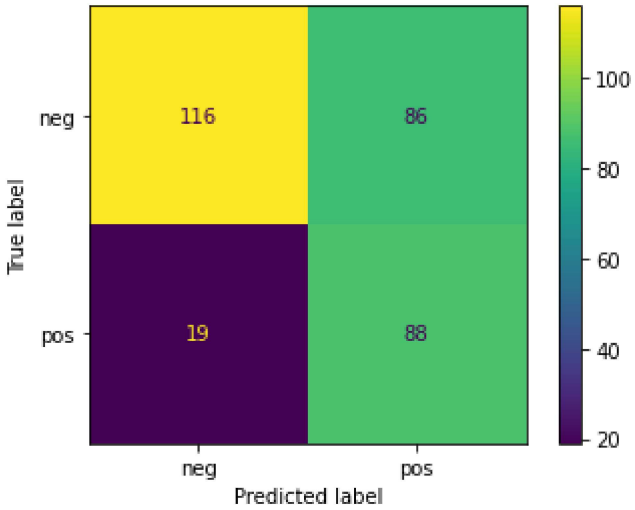
Grubhub Modeling

In [67]:
```
with open ('C:/Users/edwardsrk/final_proj/yelp_reviews_tensorflow/data/pickle
    model_sgd = pickle.load(model_p)
#reading in pickled model
```

In [69]:
```
grub_3.head()
X = grub_3.clean
y = grub_3.tag
#splitting the 3 star review data into chunks ot predict on
```

In [81]:
```
#model_sgd.fit(X, y)
#score = model_sgd.score(X,y)
#print("Test score: {0:.2f} %".format(100 * score))
#test_preds = model_sgd.predict(X)
#test_preds
sgd_acc = accuracy_score(y, test_preds)

plot_confusion_matrix(model_sgd, X, y);
print("Mean Accuracy:", cross_val_score(model_sgd, X, y).mean())
```

```
Mean Accuracy: 0.7606028556319406
```



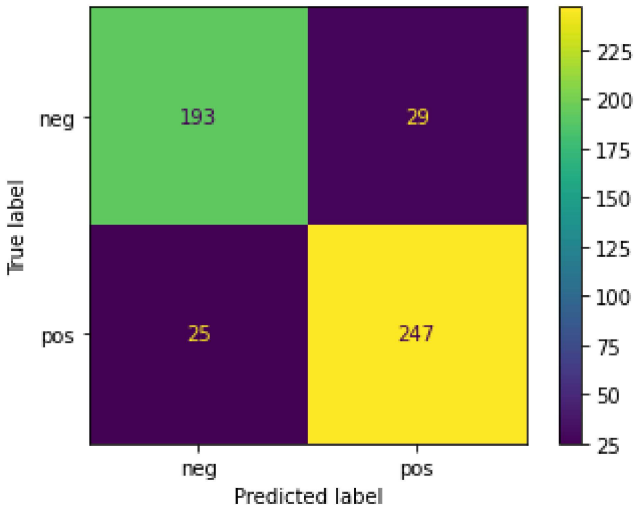
In [ ]:
```
Do to the poor performance of the model above I am going to train a my existi
```

In [62]:
```
data = grub['clean']
target = grub['tag']
#separate features from target
X_train, X_test, y_train, y_test = train_test_split(data,
                                                    target,
                                                    random_state=42,
                                                    test_size=0.25)

X_t, X_val, y_t, y_val = train_test_split(X_train, y_train,
                                          test_size=0.25, random_state=42)
# Secondary train-test split, validation set
```

This mdoel does better on the data without the three star data included in it but does just about the same when tested on the three star data as the original model.

In [64]:
```
sgd_pipe_tuned_gb = Pipeline(steps =[('tf', TfidfVectorizer(max_df = 0.75, ng
                                     ('sgd', SGDClassifier(alpha = 0.001, loss =

sgd_pipe_tuned_gb.fit(X_train, y_train)

test_preds = sgd_pipe_tuned_gb.predict(X_test)
sgd_acc = accuracy_score(y_test, test_preds)

plot_confusion_matrix(sgd_pipe_tuned_gb, X_test, y_test);
print("Mean Accuracy:", cross_val_score(sgd_pipe_tuned_gb, X_test, y_test).me
```

```
Mean Accuracy: 0.8481962481962482
```



Once again, taking a look at the features highly correlated with the negative class you can see they are very different than the negative features from the original model.

In [77]:
```
features = sgd_pipe_tuned_gb.steps[0][1].get_feature_names()
feat_values = sgd_pipe_tuned_gb[1].coef_

c = {'features' : features}
feats = pd.DataFrame(data = c)
feats['values'] = feat_values[0]

sorted_feats = feats.sort_values(by='values')
sorted_feats
```

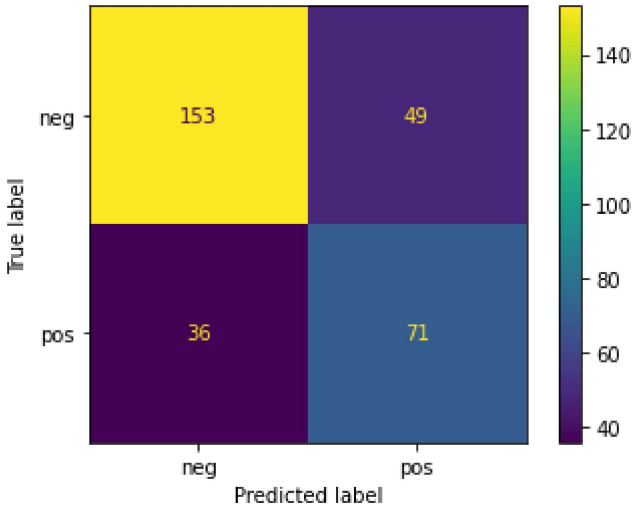
Out[77]:

	features	values
139	bad	-1.965281
1815	miss	-1.861295
393	cold	-1.693209
4347	terrible	-1.286961
108	ask	-1.271601
...	...	...
1068	good	1.787819
177	best	1.913286
1453	love	2.102992
530	delicious	2.316490
1118	great	2.412664

5000 rows × 2 columns

In [80]:
```
sgd_pipe_tuned_gb.fit(X_train, y_train)

#score = sgd_pipe_tuned_gb.score(X,y)
#print("Test score: {0:.2f} %".format(100 * score))

test_preds = sgd_pipe_tuned_gb.predict(X)
sgd_acc = accuracy_score(y, test_preds)

plot_confusion_matrix(sgd_pipe_tuned_gb, X, y);
print("Mean Accuracy:", cross_val_score(sgd_pipe_tuned_gb, X, y).mean())
```

```
Mean Accuracy: 0.7573241671073505
```



Next Steps

```
Next steps:
-Making a NER model to create an entitty distinction between Grubhub and the
actual business attached to the orders
-Gathering more data to input into another model for better predictions
-Figure out how to deal with the opposite distribution
-Deploy an app in Flask that you can type a review into
```