Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews (https://www.kaggle.com/snap/amazon-fine-food-reviews)

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/ (https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/)

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective:

Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative?

[Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [6]:

```
!pip install kaggle
from google.colab import files
files.upload()
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.6/di
st-packages (1.5.2)
Requirement already satisfied: urllib3<1.23.0,>=1.15 in /usr/local/l
ib/python3.6/dist-packages (from kaggle) (1.22)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6
/dist-packages (from kaggle) (1.11.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.6/d
ist-packages (from kaggle) (2018.11.29)
Requirement already satisfied: python-dateutil in /usr/local/lib/pyt
hon3.6/dist-packages (from kaggle) (2.5.3)
Requirement already satisfied: requests in /usr/local/lib/python3.6/
dist-packages (from kaggle) (2.18.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist
-packages (from kaggle) (4.28.1)
Requirement already satisfied: python-slugify in /usr/local/lib/pyth
on3.6/dist-packages (from kaggle) (2.0.1)
Requirement already satisfied: idna<2.7,>=2.5 in /usr/local/lib/pyth
on3.6/dist-packages (from requests->kaggle) (2.6)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/l
ib/python3.6/dist-packages (from requests->kaggle) (3.0.4)
Requirement already satisfied: Unidecode>=0.04.16 in /usr/local/lib/
python3.6/dist-packages (from python-slugify->kaggle) (1.0.23)
```

Choose Files no files selected

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[6]:

```
{'kaggle.json': b'{"username":"rohitbohra2994","key":"162fd3c566c99c
9e6047f8248ccf8a9e"}'}
```

In [9]:

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/


!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d snap/amazon-fine-food-reviews

!ls
```

```
401 - Unauthorized
kaggle.json  sample_data
```

In [10]:

```
!unzip amazon-fine-food-reviews.zip
```

```
unzip:  cannot find or open amazon-fine-food-reviews.zip, amazon-fin
e-food-reviews.zip.zip or amazon-fine-food-reviews.zip.ZIP.
```

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

In [2]:

```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 5
# you can change the number to any other number based on your computin

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Sc
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score

# Give reviews with Score>3 a positive rating(1), and reviews with a sc
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (525814, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDen
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	

In [3]:

```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [4]:

```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:

```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUI
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	

In [6]:

```
display['COUNT(*)'].sum()
```

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [7]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDe
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=T
```

In [9]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time
final.shape
```

Out[9]:
```
(364173, 10)
```

In [10]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[10]:
```
69.25890143662969
```

In [11]:

```
#sorting of data
final['Time'] = pd.to_datetime(final['Time'])
# Sort by time
final = final.sort_values(by='Time')

print(final.shape)
print(final['Score'].value_counts())

print(final['Text'][1])
```

```
(364173, 10)
1    307063
0     57110
Name: Score, dtype: int64
Product arrived labeled as Jumbo Salted Peanuts...the peanuts were a
ctually small sized unsalted. Not sure if this was an error or if th
e vendor intended to represent the product as "Jumbo".
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [12]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDe
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

In [13]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [14]:

```
#Before starting the next phase of preprocessing lets see the number o
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(364171, 10)
```

Out[14]:

```
1    307061
0     57110
Name: Score, dtype: int64
```

In [15]:

```
final.head()
```

Out[15]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulr
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	
417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	
346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	
417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [16]:
```python
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
this witty little book makes my son laugh at loud. i recite it in th
e car as we're driving along and he always can sing the refrain. he'
s learned about whales, India, drooping roses:  i love all the new w
ords this book  introduces and the silliness of it all.  this is a c
lassic book i am  willing to bet my son will STILL be able to recite
from memory when he is  in college
==================================================
I finally ordered a couple products from this seller for myself(not
as gifts) and I am really happy. This Jade Bonsai is really cool and
it arrived fast and in perfect condition. It's in my living room and
I get tons of compliments. It's already grown some too and the pot i
t came in is really nice, looks expensive! Much bigger than I though
t it would be even.  Thanks again!!
==================================================
I bought some of this tea when I was in Seattle and I have been dyin
g to get more.  It really is the best tea I have ever had.  It is gr
eat hot or cold.
==================================================
I would prefer freshly made brown rice, but that takes a long time t
o make and isn't easy. This makes it convenient, and takes all the g
uess work out of making it. I generally have been buying frozen orga
nic brown rice, but that takes up lots of freezer space. The fact th
at this is easy to store at room temperature is a big plus. I'll be
buying more.
==================================================
```

In [17]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/4
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in th
e car as we're driving along and he always can sing the refrain. he'
s learned about whales, India, drooping roses:  i love all the new w
ords this book  introduces and the silliness of it all.  this is a c
lassic book i am  willing to bet my son will STILL be able to recite
from memory when he is  in college
```

In [18]:

```python
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-ho
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
this witty little book makes my son laugh at loud. i recite it in th
e car as we're driving along and he always can sing the refrain. he'
s learned about whales, India, drooping roses:  i love all the new w
ords this book  introduces and the silliness of it all.  this is a c
lassic book i am  willing to bet my son will STILL be able to recite
from memory when he is  in college
==================================================
I finally ordered a couple products from this seller for myself(not
as gifts) and I am really happy. This Jade Bonsai is really cool and
it arrived fast and in perfect condition. It's in my living room and
I get tons of compliments. It's already grown some too and the pot i
t came in is really nice, looks expensive! Much bigger than I though
t it would be even.  Thanks again!!
==================================================
I bought some of this tea when I was in Seattle and I have been dyin
g to get more.  It really is the best tea I have ever had.  It is gr
eat hot or cold.
==================================================
I would prefer freshly made brown rice, but that takes a long time t
o make and isn't easy. This makes it convenient, and takes all the g
uess work out of making it. I generally have been buying frozen orga
nic brown rice, but that takes up lots of freezer space. The fact th
at this is easy to store at room temperature is a big plus. I'll be
buying more.
```

In [19]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [20]:

```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
I bought some of this tea when I was in Seattle and I have been dyin
g to get more.  It really is the best tea I have ever had.  It is gr
eat hot or cold.
==================================================
```

In [21]:

```
#remove words with numbers python: https://stackoverflow.com/a/1808237
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in th
e car as we're driving along and he always can sing the refrain. he'
s learned about whales, India, drooping roses:  i love all the new w
ords this book  introduces and the silliness of it all.  this is a c
lassic book i am  willing to bet my son will STILL be able to recite
from memory when he is  in college
```

In [22]:

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
I bought some of this tea when I was in Seattle and I have been dyin
g to get more It really is the best tea I have ever had It is great
hot or cold
```

In [23]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in t

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ou
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves'
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'thi
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'ha
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'i
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'ho
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', '
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'd
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
            'won', "won't", 'wouldn', "wouldn't"])
```

In [24]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 364171/364171 [03:08<00:00, 1930.37it/s]
```

In [25]:
```
# Random sampling
data = final.head(100000)

data_preprocessed_reviews=preprocessed_reviews[0:100000]
```

In [26]:
```
print("The size of sampled data is ",data.shape)
print("The size of sampled data is ",len(data_preprocessed_reviews))
```

```
The size of sampled data is  (100000, 10)
The size of sampled data is  100000
```

[4] Featurization

[4.1] BAG OF WORDS

In [27]:

```
# Spliting into Train and test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data_preprocessed_:
```

In [28]:

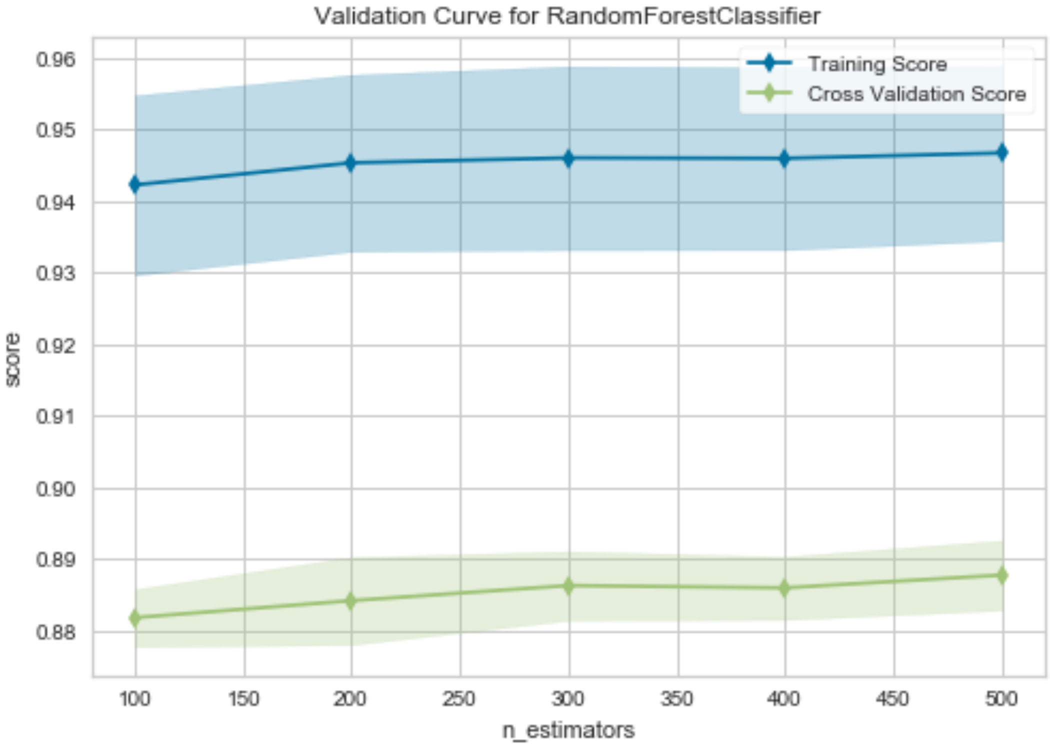
```
#Bag of words
count_vect = CountVectorizer(max_features=2000, min_df=20)

x_train = count_vect.fit_transform(x_train)



x_test = count_vect.transform(x_test)
```

In [29]:

```
om sklearn.ensemble import RandomForestClassifier
om sklearn.model_selection import TimeSeriesSplit
om sklearn.model_selection import GridSearchCV
om yellowbrick.model_selection import ValidationCurve

se_learners = [100,200,300,400,500]
pth = (range(1,20,4))
ram_grid={"n_estimators":base_learners,"max_depth":depth}
cv = TimeSeriesSplit(n_splits=5)
 = RandomForestClassifier(max_features='sqrt', class_weight="balanced")
v = GridSearchCV(rf, param_grid,scoring='roc_auc',cv=tscv,n_jobs=-1,ver
v.fit(x_train, y_train)
_scores = gsv.cv_results_['mean_test_score']
int("Model with best parameters :\n",gsv.best_estimator_)
int("Best Score: %.2f%%"%(gsv.best_score_*100))
int("Best HyperParameter: ",gsv.best_params_)
timal_n_estimators = gsv.best_estimator_.n_estimators
timal_max_depth = gsv.best_estimator_.max_depth

alidation Curve
z = ValidationCurve(gsv.best_estimator_, param_name="n_estimators",para
z.fit(x_train, y_train)
z.poof()
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent w
orkers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 125 out of 125 | elapsed:  8.2min finish
```

```
ed

Model with best parameters :
 RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=13, max_features='sqrt',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=500, n_jobs=None, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
Best Score: 88.69%
Best HyperParameter:  {'max_depth': 13, 'n_estimators': 500}
```



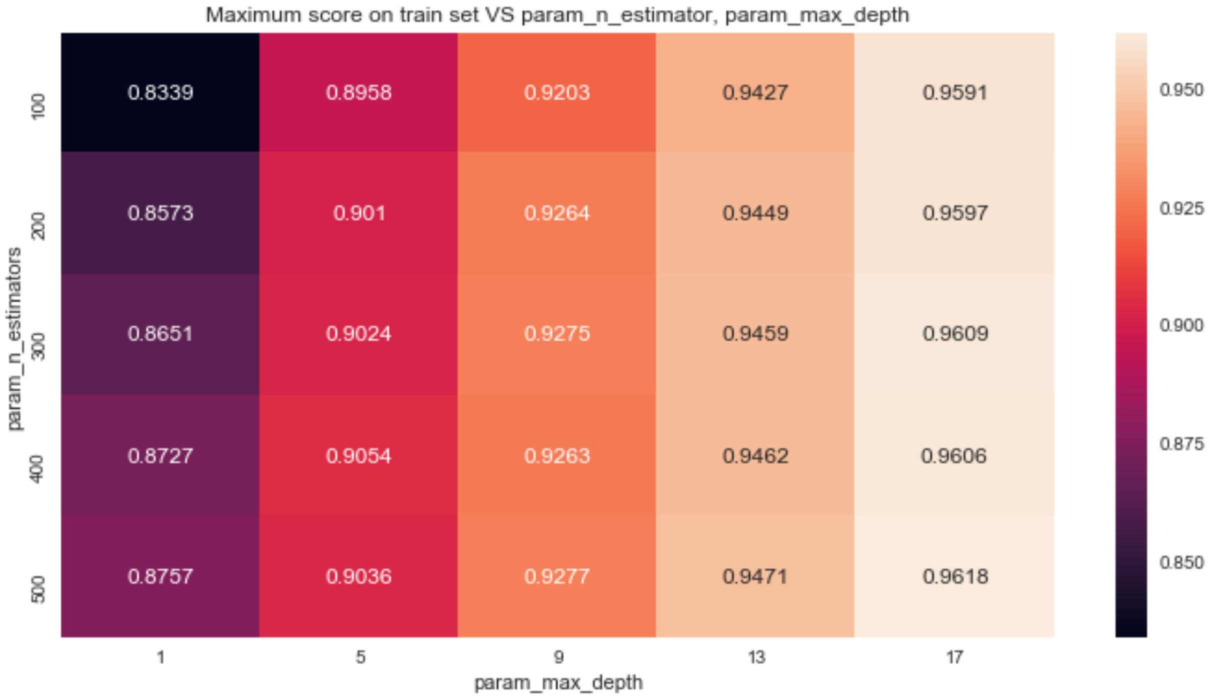
In [30]:
```
gsv.best_estimator_
```

Out[30]:
```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=13, max_features='sqrt',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=500, n_jobs=None, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
```

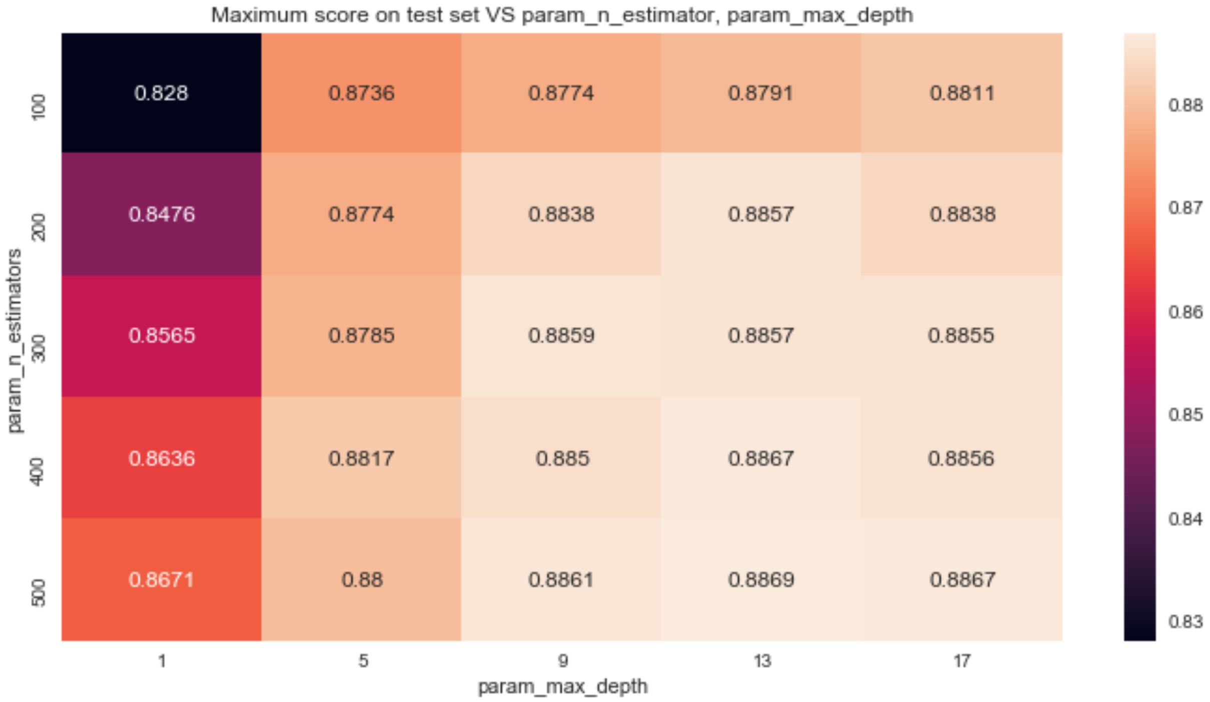
In [31]:
```
gsv.cv_results_
```

```
False,
                        False],
         fill_value='?',
               dtype=object),
 'param_n_estimators': masked_array(data=[100, 200, 300, 400, 500, 1
00, 200, 300, 400, 500, 100,
                        200, 300, 400, 500, 100, 200, 300, 400, 500, 100
, 200,
                        300, 400, 500],
                 mask=[False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False, False, False, False, False, False, False,
False,
                        False],
         fill_value='?',
               dtype=object),
 'params': [{'max_depth': 1, 'n_estimators': 100},
  {'max depth': 1, 'n estimators': 200},
```

In [34]:
```
df_gridsearch = pd.DataFrame(gsv.cv_results_)
max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_de
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Maximum score on train set VS param_n_estimator, param_max_de
fmt = 'png'
sns.heatmap(max_scores.mean_train_score, annot=True, fmt='.4g');
plt.title(title);
```



In [32]:
```
df_gridsearch = pd.DataFrame(gsv.cv_results_)
max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_de
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Maximum score on test set VS param_n_estimator, param_max_dep
fmt = 'png'
sns.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g');
plt.title(title);
```

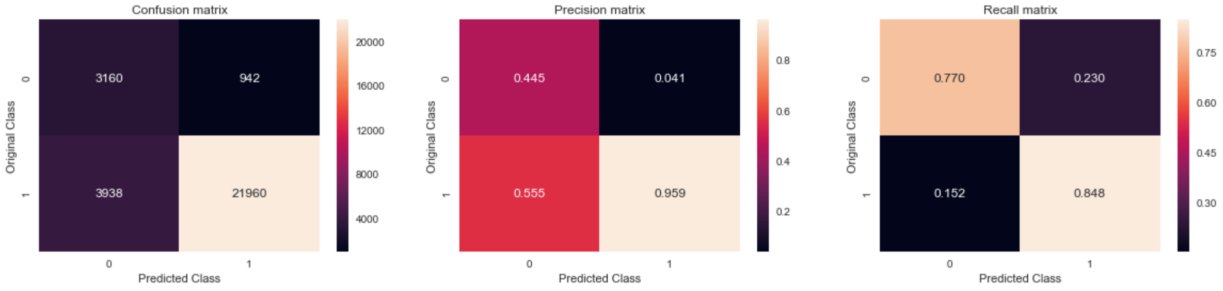


In [35]:

```
# This function plots the confusion, precision and recall matrices
def plot_confusion_matrix(x_test, y_pred):
    C = confusion_matrix(x_test, y_pred)

    A =(((C.T)/(C.sum(axis=1))).T)
    B =(C/C.sum(axis=0))
    plt.figure(figsize=(20,4))

    labels = [0,1]

    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, fmt="d", xticklabels=labels, yticklabel
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, fmt=".3f", xticklabels=labels, yticklab
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    # representing B in heatmap format
    sns.heatmap(A, annot=True, fmt=".3f", xticklabels=labels, yticklab
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

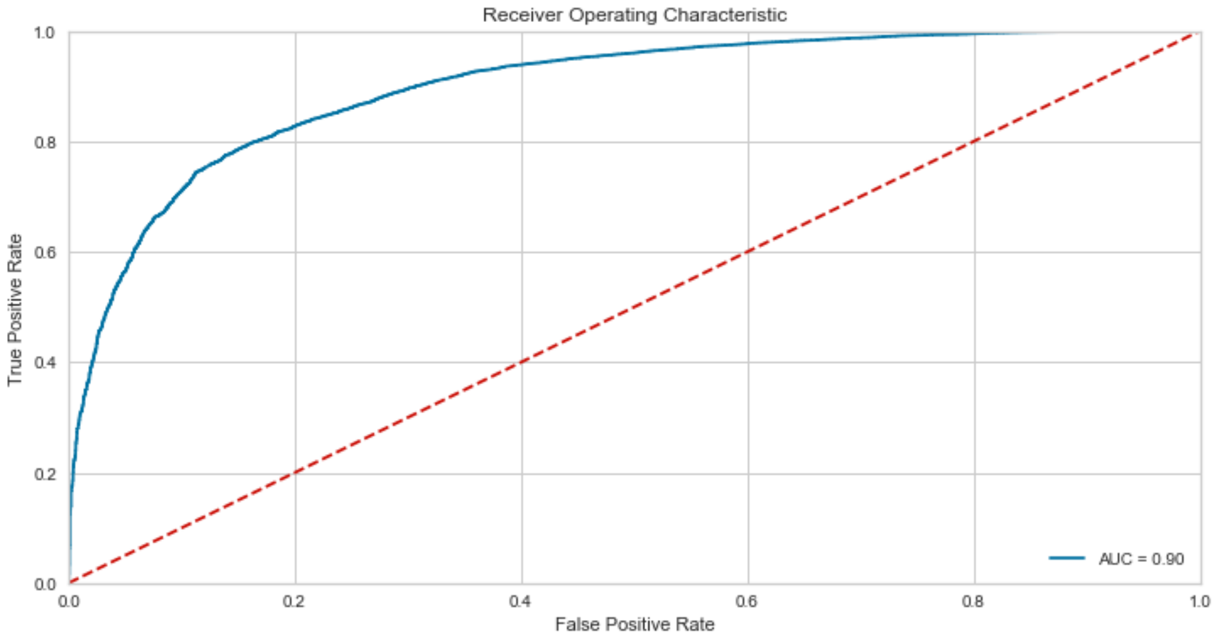
In [36]:
```
ting Accuracy on Test data
rics
 sklearn.metrics import accuracy_score
 sklearn.metrics import confusion_matrix
 sklearn.metrics import precision_score
 sklearn.metrics import recall_score
 sklearn.metrics import f1_score
 sklearn.metrics import roc_curve, auc
 sklearn.metrics import roc_auc_score



ting Accuracy on Test data
t("The optimal value of n_estimators is : ",optimal_n_estimators)
t("The optimal value of max_depth is : ",optimal_max_depth)
 RandomForestClassifier(n_estimators=optimal_n_estimators, max_depth=op
it(x_train,y_train)
ed = rf.predict(x_test)
ob = rf.predict_proba(x_test)


t("ROC_AUC on test set: %0.3f" % roc_auc_score(y_test, y_prob[:,1]))
t("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
t("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
t("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
t("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
_confusion_matrix(y_test, y_pred)
```

```
The optimal value of n_estimators is :  500
The optimal value of max_depth is :  13
ROC_AUC on test set: 0.898
Accuracy on test set: 83.733%
Precision on test set: 0.959
Recall on test set: 0.848
F1-Score on test set: 0.900
```

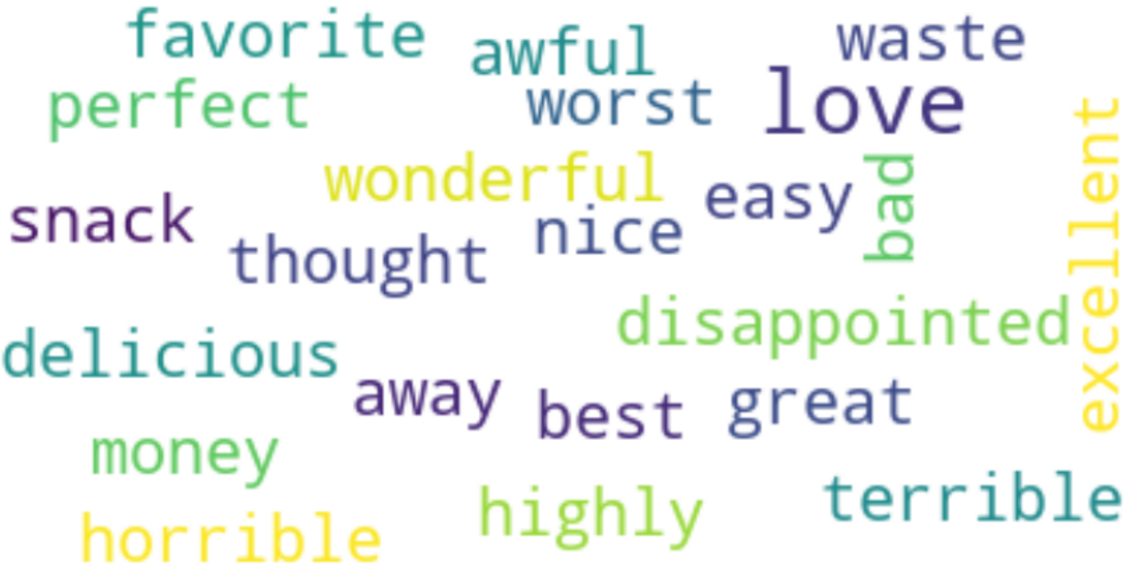


In [39]:

```
preds = y_prob[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
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 [40]:
```
#https://www.datacamp.com/community/tutorials/wordcloud-python
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt

feat_importances=rf.feature_importances_
feat_names=count_vect.get_feature_names()

# Sort feature importances in descending order
indices = np.argsort(feat_importances)[::-1][:25]
a=np.take(feat_names,indices)
def words(X):
    comment_words=' '
    for words in X:
        comment_words = comment_words + words + ' '
    return comment_words
a=words(a)

#Word Cloud
wc = WordCloud(max_font_size=30, max_words=100, background_color="whit
wc.generate(a)
plt.figure(figsize=[15,10])
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

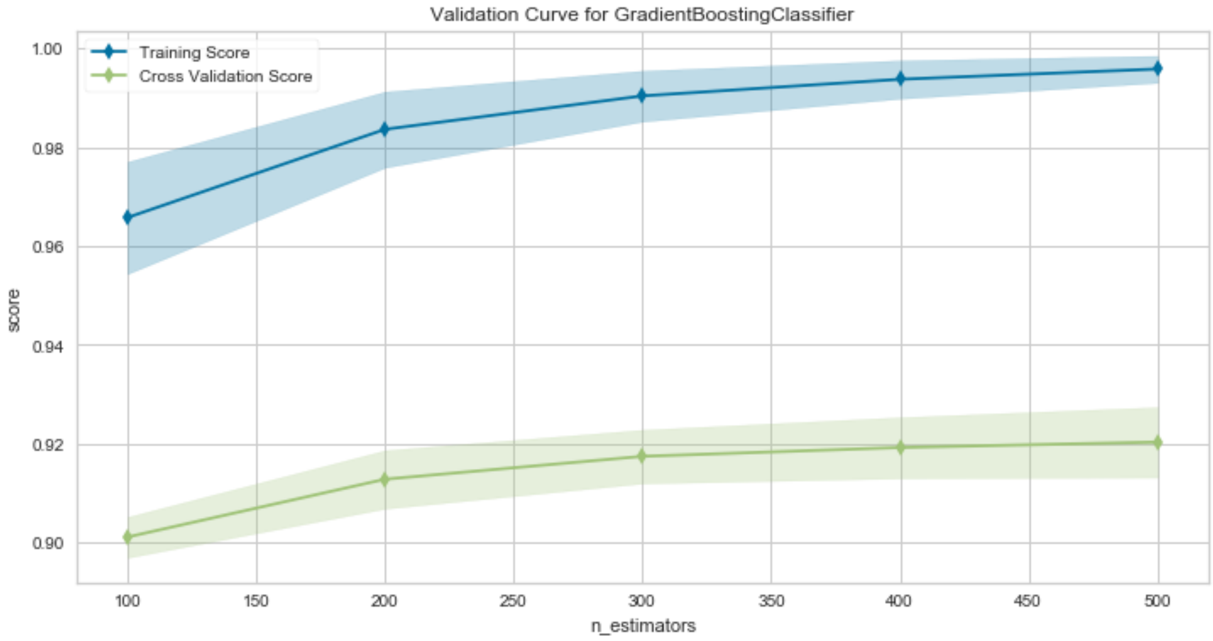


In [41]:
```
om sklearn.ensemble import GradientBoostingClassifier
om sklearn.model_selection import GridSearchCV
om yellowbrick.model_selection import ValidationCurve

se_learners = [100,200,300,400,500]
pth = (range(1,20,4))
arning_rate = [0.01,0.05,0.1,0.2,0.3]

ram_grid = {'n_estimators': base_learners, 'max_depth':depth, 'learning
cv = TimeSeriesSplit(n_splits=5)
```

```
:v = TimeSeriesSplit(n_splits=5)
) = GradientBoostingClassifier(max_features='sqrt')
;v = GridSearchCV(gb, param_grid,scoring='roc_auc',cv=tscv,n_jobs=-1,ver
;v.fit(x_train, y_train)
_scores = gsv.cv_results_['mean_test_score']
int("Model with best parameters :\n",gsv.best_estimator_)
int("Best HyperParameter: ",gsv.best_params_)
int("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
timal_n_estimators = gsv.best_estimator_.n_estimators
timal_max_depth = gsv.best_estimator_.max_depth
timal_learning_rate = gsv.best_estimator_.learning_rate

'alidation Curve
z = ValidationCurve(gsv.best_estimator_, param_name="n_estimators",para
z.fit(x_train, y_train)
z.poof()
```

```
Fitting 5 folds for each of 125 candidates, totalling 625 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent w
orkers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed: 23.1min
/Users/rohitbohra/anaconda3/lib/python3.6/site-packages/sklearn/exte
rnals/joblib/externals/loky/process_executor.py:706: UserWarning: A
worker stopped while some jobs were given to the executor. This can
be caused by a too short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
/Users/rohitbohra/anaconda3/lib/python3.6/site-packages/sklearn/exte
rnals/joblib/externals/loky/process_executor.py:706: UserWarning: A
worker stopped while some jobs were given to the executor. This can
be caused by a too short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
/Users/rohitbohra/anaconda3/lib/python3.6/site-packages/sklearn/exte
rnals/joblib/externals/loky/process_executor.py:706: UserWarning: A
worker stopped while some jobs were given to the executor. This can
be caused by a too short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed: 57.6min
[Parallel(n_jobs=-1)]: Done 625 out of 625 | elapsed: 86.9min finish
ed

Model with best parameters :
 GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=9,
              max_features='sqrt', max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=400,
              n_iter_no_change=None, presort='auto', random_state=No
ne,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
Best HyperParameter:  {'learning_rate': 0.1, 'max_depth': 9, 'n_esti
mators': 400}
```

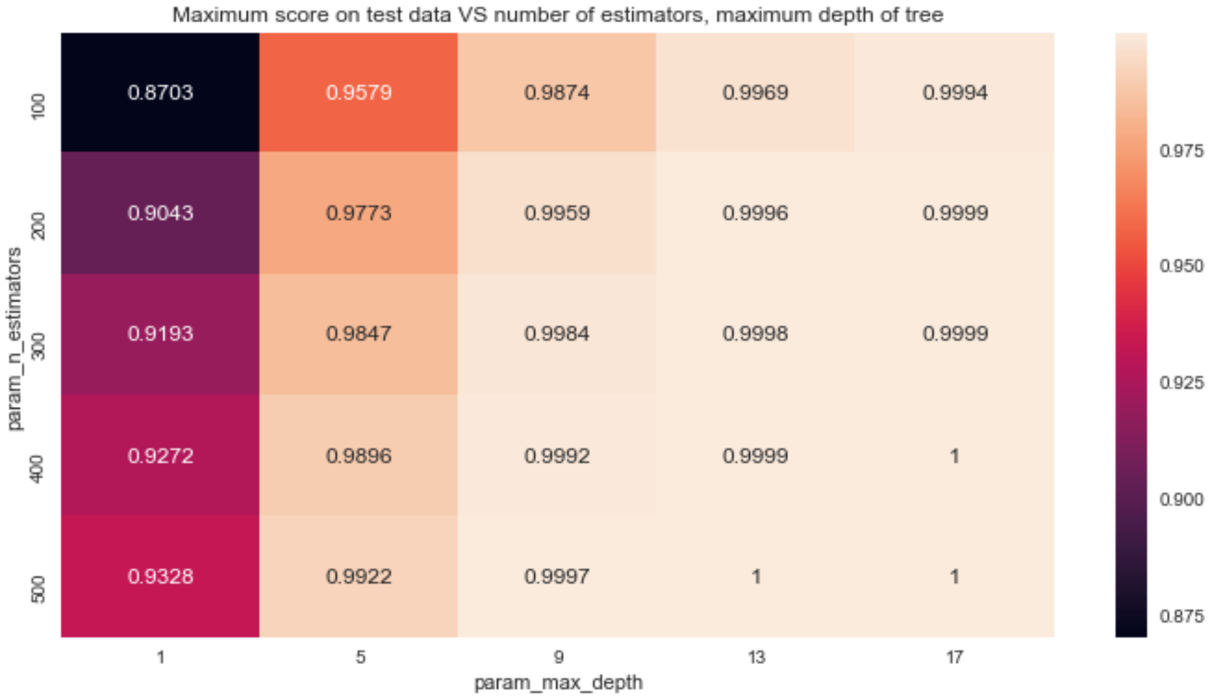
```
Best Accuracy: 92.04%
```



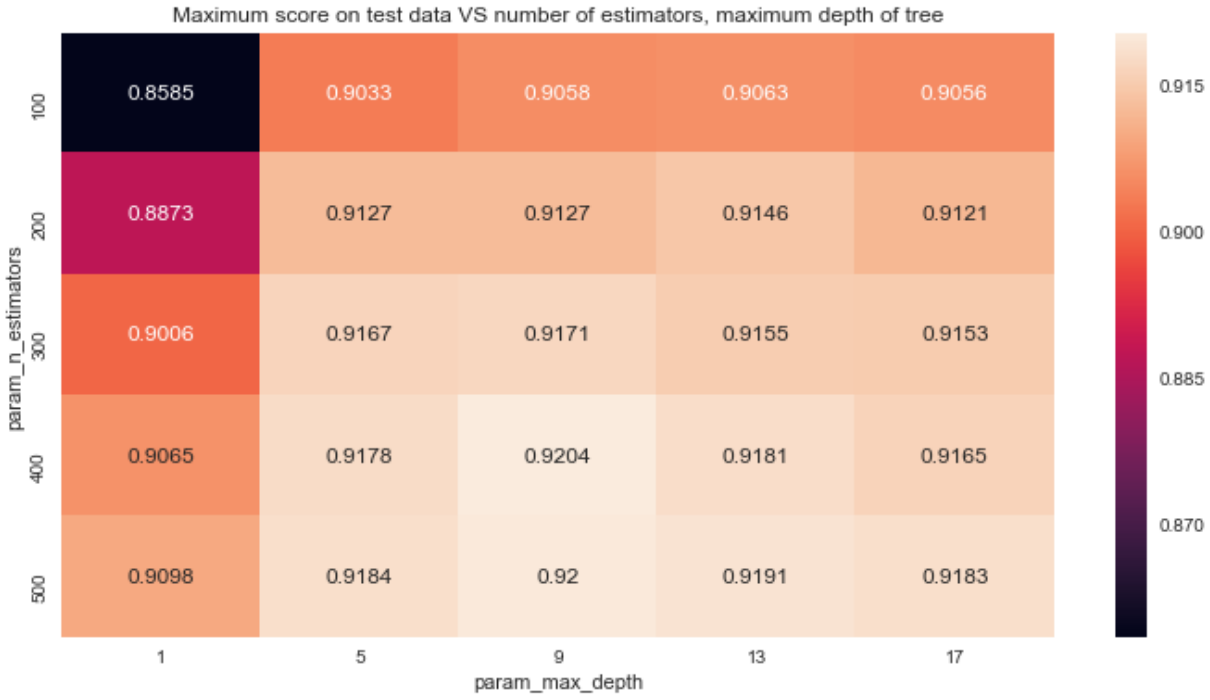
In [42]:
```
gsv.best_estimator_
```

Out[42]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=9,
                           max_features='sqrt', max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=400,
                           n_iter_no_change=None, presort='auto', random_state=No
ne,
                           subsample=1.0, tol=0.0001, validation_fraction=0.1,
                           verbose=0, warm_start=False)
```

In [43]:
```
df_gridsearch = pd.DataFrame(gsv.cv_results_)
max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_de
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Maximum score on test data VS number of estimators, maximum d
fmt = 'png'
sns.heatmap(max_scores.mean_train_score, annot=True, fmt='.4g');
plt.title(title);
```

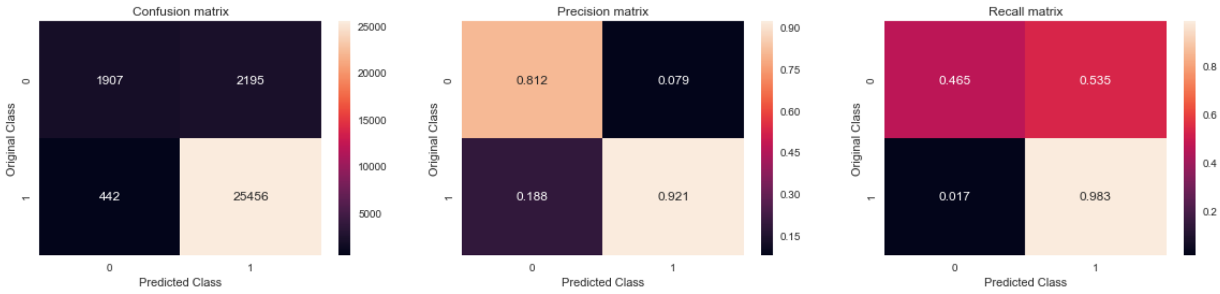


In [44]:
```
df_gridsearch = pd.DataFrame(gsv.cv_results_)
max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_de
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Maximum score on test data VS number of estimators, maximum d
fmt = 'png'
sns.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g');
plt.title(title);
```



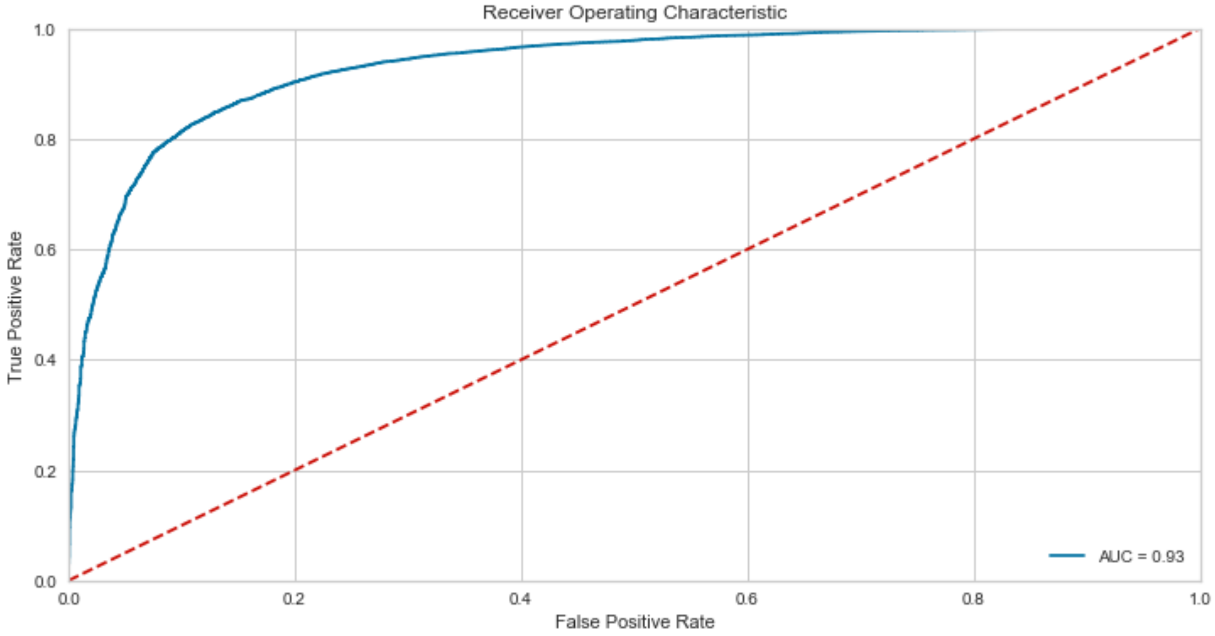
In [45]:
```
sting Accuracy on Test data
trics
n sklearn.metrics import accuracy_score
n sklearn.metrics import confusion_matrix
n sklearn.metrics import precision_score
n sklearn.metrics import recall_score
n sklearn.metrics import f1_score
n sklearn.metrics import roc_curve, auc
n sklearn.metrics import roc_auc_score



sting Accuracy on Test data
nt("The optimal value of n_estimators is : ",optimal_n_estimators)
nt("The optimal value of max_depth is : ",optimal_max_depth)

= GradientBoostingClassifier(n_estimators=optimal_n_estimators, max_dept
fit(x_train,y_train)
red = gb.predict(x_test)
rob = gb.predict_proba(x_test)


nt("ROC_AUC on test set: %0.3f" % roc_auc_score(y_test, y_prob[:,1]))
nt("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100)
nt("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
nt("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
nt("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
t_confusion_matrix(y_test, y_pred)
```

```
The optimal value of n_estimators is :  400
The optimal value of max_depth is :  9
ROC_AUC on test set: 0.932
Accuracy on test set: 91.210%
Precision on test set: 0.921
Recall on test set: 0.983
F1-Score on test set: 0.951
```



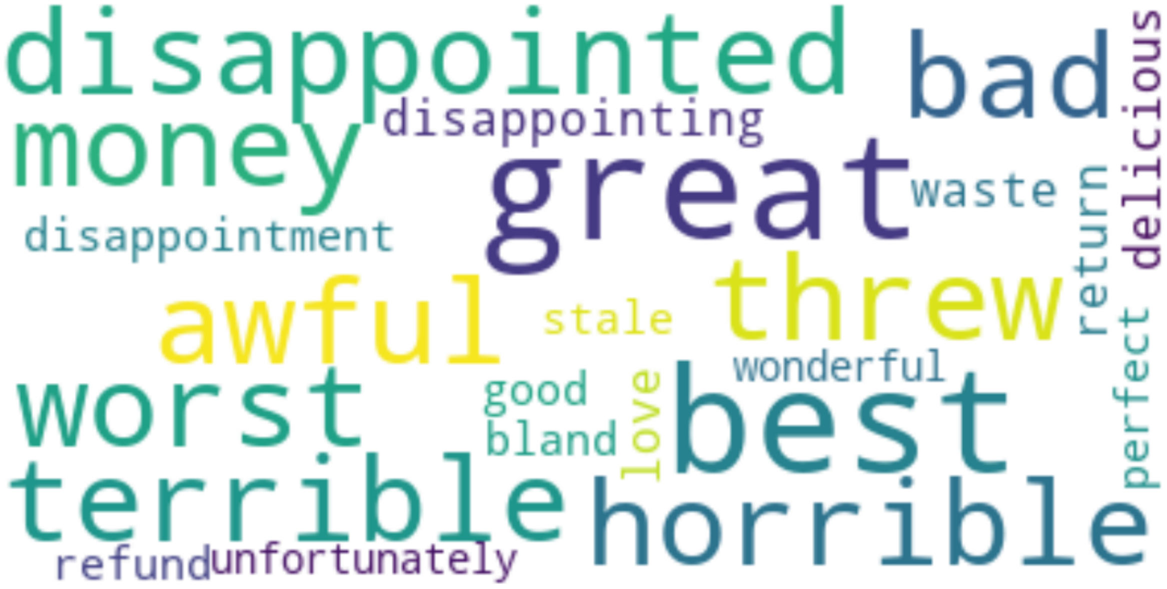
In [46]:

```
preds = y_prob[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
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 [47]:

```
#https://www.datacamp.com/community/tutorials/wordcloud-python
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt

importances=gb.feature_importances_
feat_names=count_vect.get_feature_names()

# Sort feature importances in descending order
indices = np.argsort(importances)[::-1][:25]
a=np.take(feat_names,indices)
def words(X):
    comment_words=' '
    for words in X:
        comment_words = comment_words + words + ' '
    return comment_words
a=words(a)

#Word Cloud
wc = WordCloud(max_font_size=50, max_words=100, background_color="white
wc.generate(a)
plt.figure(figsize=[20,10])
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```



[4.3] TF-IDF

In [48]:

```
from sklearn import preprocessing
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer


# Spliting into Train and test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data_preprocessed_
tfidf = TfidfVectorizer(ngram_range=(1,2), max_features=2000, min_df=2

#preparing the train data
x_train = tfidf.fit_transform(x_train)
#Normalize Data
x_train = preprocessing.normalize(x_train)
print("Train Data Size: ",x_train.shape)
print("the type of count vectorizer for train data is ",type(x_train))
print("the shape of train data is ",x_train.get_shape())
print("the number of unique words including both unigrams and bigrams 
print("some sample features(unique words) ",tfidf.get_feature_names()[

#preparing the test data
x_test = tfidf.transform(x_test)
#Normalize Data
x_test = preprocessing.normalize(x_test)
print("Test Data Size: ",x_test.shape)
print("the type of count vectorizer for test data is ",type(x_test))
print("the shape of test data is ",x_test.get_shape())
print("the number of unique words including both unigrams and bigrams 
```

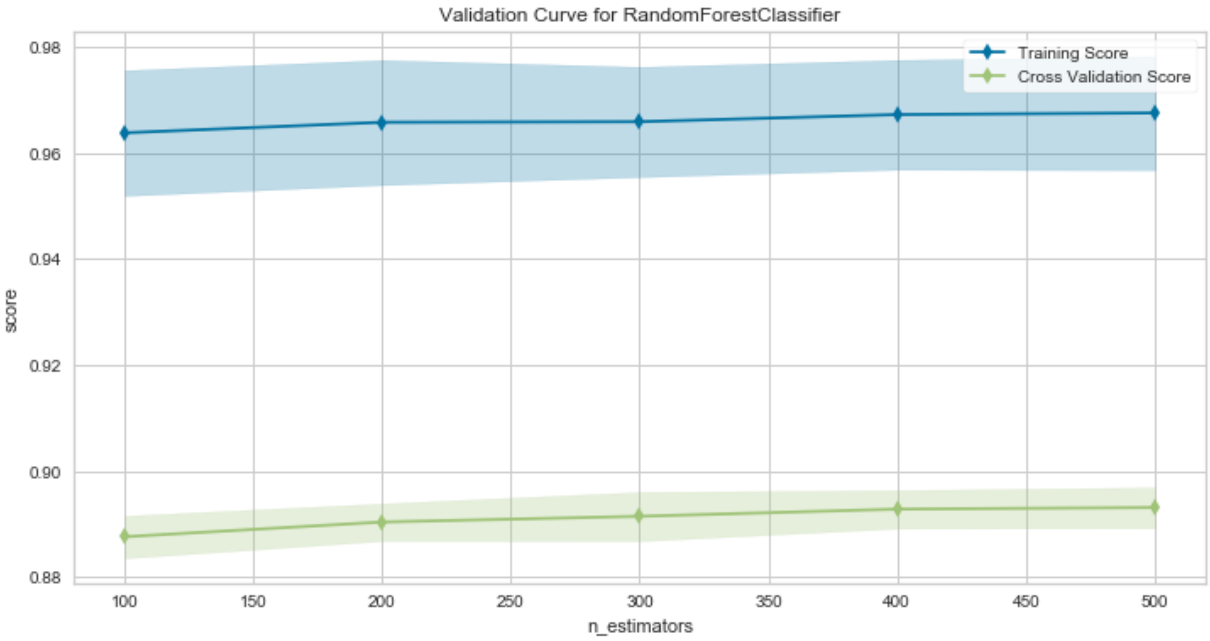
```
Train Data Size:  (70000, 2000)
the type of count vectorizer for train data is  <class 'scipy.sparse
.csr.csr_matrix'>
the shape of train data is  (70000, 2000)
the number of unique words including both unigrams and bigrams for t
rain data is  2000
some sample features(unique words)  ['able', 'able find', 'absolute'
, 'absolutely', 'absolutely delicious', 'absolutely love', 'accordin
g', 'acid', 'across', 'actual']
Test Data Size:  (30000, 2000)
the type of count vectorizer for test data is  <class 'scipy.sparse.
csr.csr_matrix'>
the shape of test data is  (30000, 2000)
the number of unique words including both unigrams and bigrams for t
est data is  2000
```

In [49]:

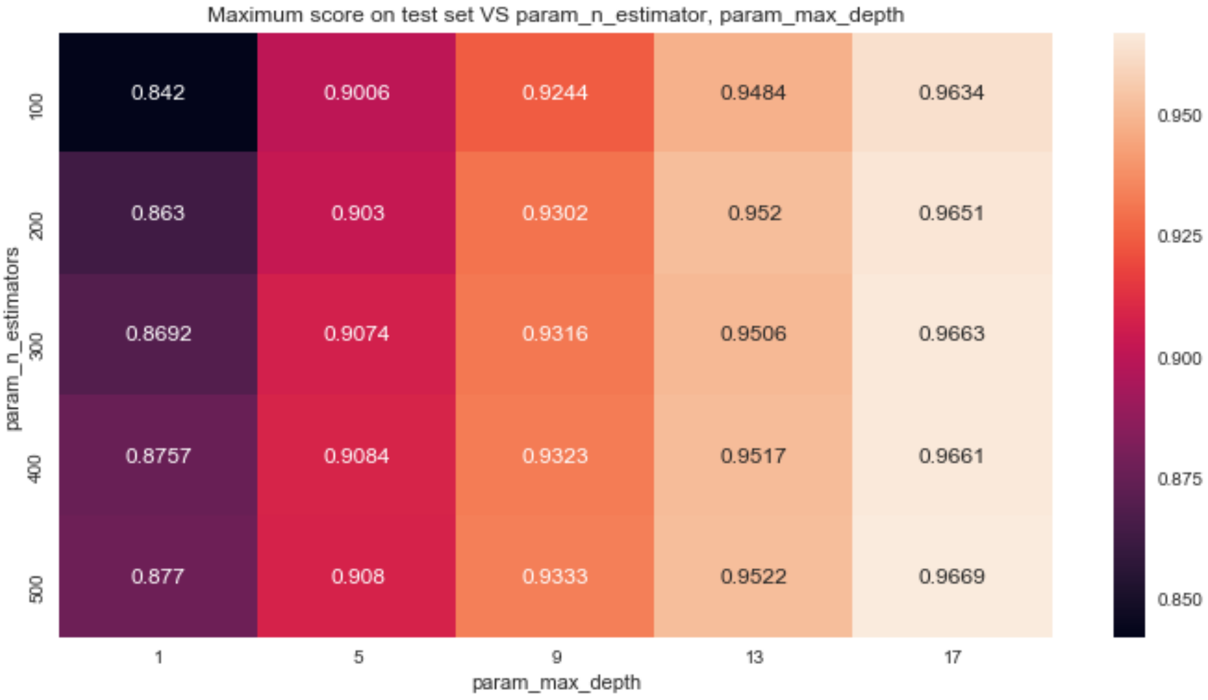
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import GridSearchCV
from yellowbrick.model_selection import ValidationCurve

base_learners = [100,200,300,400,500]
depth = (range(1,20,4))
```

```
param_grid={"n_estimators":base_learners,"max_depth":depth}
tscv = TimeSeriesSplit(n_splits=5)
rf = RandomForestClassifier(max_features='sqrt', class_weight="balance
gsv = GridSearchCV(rf, param_grid,scoring='roc_auc',cv=tscv,n_jobs=-1,
gsv.fit(x_train, y_train)
cv_scores = gsv.cv_results_['mean_test_score']
print("Model with best parameters :\n",gsv.best_estimator_)
print("Best Score: %.2f%%"%(gsv.best_score_*100))
print("Best HyperParameter: ",gsv.best_params_)
optimal_n_estimators = gsv.best_estimator_.n_estimators
optimal_max_depth = gsv.best_estimator_.max_depth

#Validation Curve
viz = ValidationCurve(gsv.best_estimator_, param_name="n_estimators",p
viz.fit(x_train, y_train)
viz.poof()
```

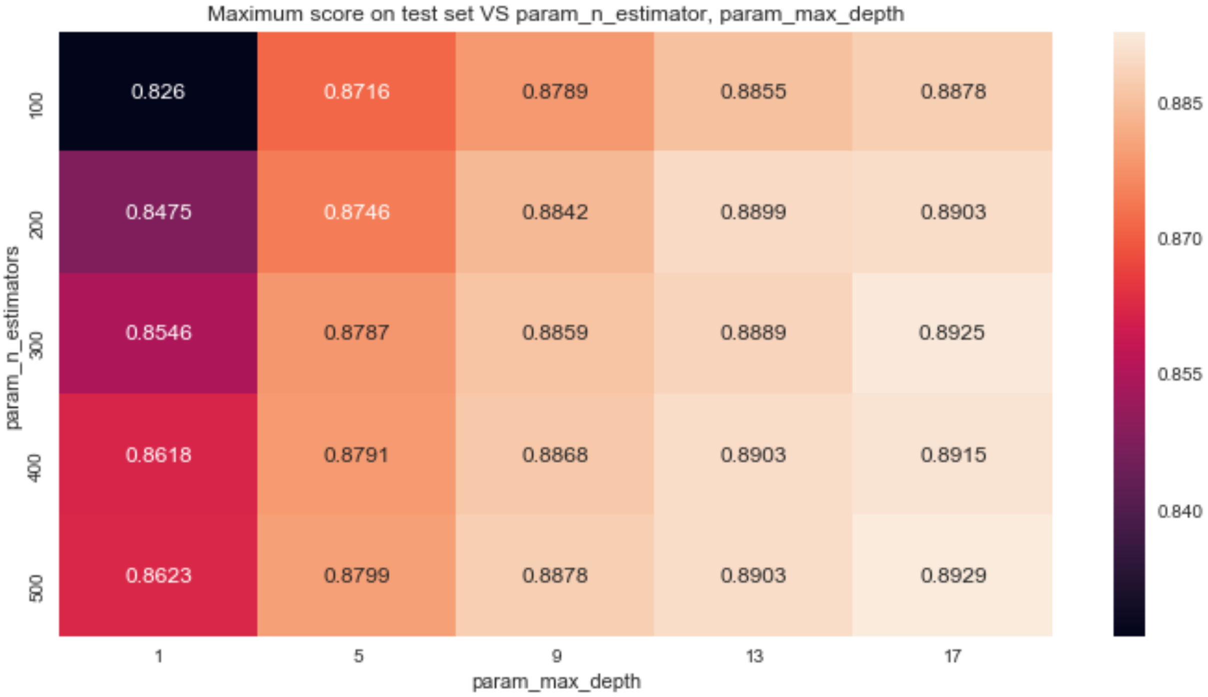
```
Fitting 5 folds for each of 25 candidates, totalling 125 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent w
orkers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 125 out of 125 | elapsed: 10.4min finish
ed

Model with best parameters :
 RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=17, max_features='sqrt',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=500, n_jobs=None, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
Best Score: 89.29%
Best HyperParameter:  {'max_depth': 17, 'n_estimators': 500}
```



In [50]:
```
df_gridsearch = pd.DataFrame(gsv.cv_results_)
max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_de
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Maximum score on test set VS param_n_estimator, param_max_dep
fmt = 'png'
sns.heatmap(max_scores.mean_train_score, annot=True, fmt='.4g');
plt.title(title);
```

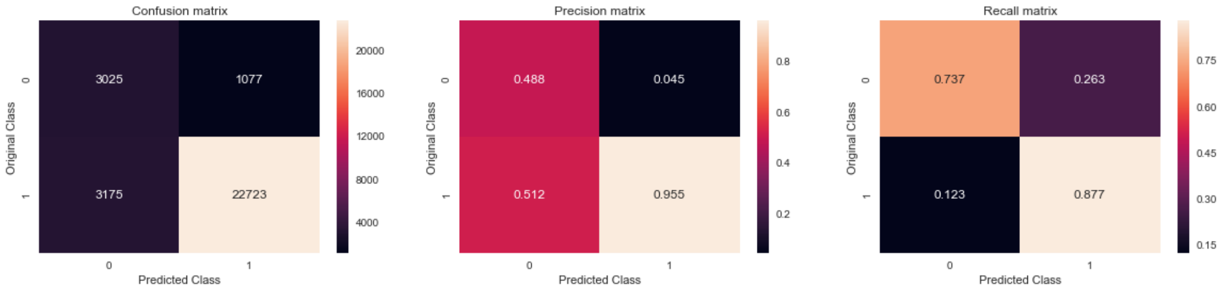


```
In [51]: df_gridsearch = pd.DataFrame(gsv.cv_results_)
         max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_de
         import matplotlib.pyplot as plt
         plt.rcParams["figure.figsize"] = (12, 6)
         title = 'Maximum score on test set VS param_n_estimator, param_max_dep
         fmt = 'png'
         sns.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g');
         plt.title(title);
```



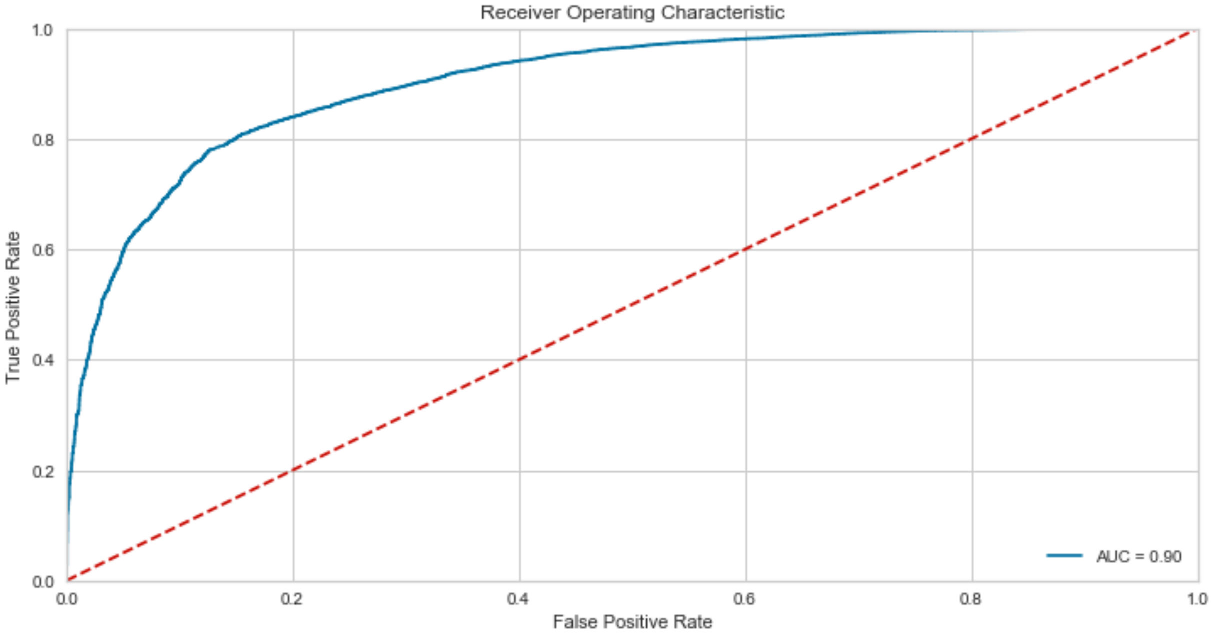
In [52]:
```
Testing Accuracy on Test data
Metrics
rom sklearn.metrics import accuracy_score
rom sklearn.metrics import confusion_matrix
rom sklearn.metrics import precision_score
rom sklearn.metrics import recall_score
rom sklearn.metrics import f1_score
rom sklearn.metrics import roc_curve, auc
rom sklearn.metrics import roc_auc_score



Testing Accuracy on Test data
rint("The optimal value of n_estimators is : ",optimal_n_estimators)
rint("The optimal value of max_depth is : ",optimal_max_depth)
f = RandomForestClassifier(n_estimators=optimal_n_estimators, max_depth
f.fit(x_train,y_train)
_pred = rf.predict(x_test)
_prob = rf.predict_proba(x_test)


rint("ROC_AUC on test set: %0.3f" % roc_auc_score(y_test, y_prob[:,1]))
rint("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*10
rint("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
rint("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
rint("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
lot_confusion_matrix(y_test, y_pred)
```

```
The optimal value of n_estimators is :  500
The optimal value of max_depth is :  17
ROC_AUC on test set: 0.904
Accuracy on test set: 85.827%
Precision on test set: 0.955
Recall on test set: 0.877
F1-Score on test set: 0.914
```



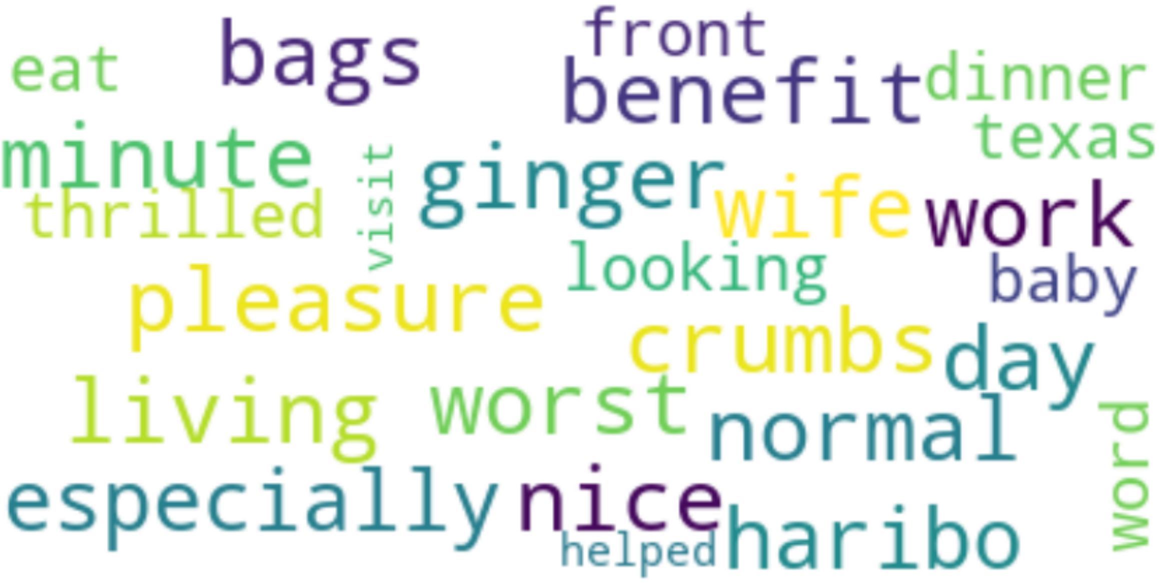
In [53]:

```
preds = y_prob[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
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 [54]:

```
#https://www.datacamp.com/community/tutorials/wordcloud-python
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt

feat_importances=rf.feature_importances_
feat_names=count_vect.get_feature_names()

# Sort feature importances in descending order
indices = np.argsort(feat_importances)[::-1][:25]
a=np.take(feat_names,indices)
def words(X):
    comment_words=' '
    for words in X:
        comment_words = comment_words + words + ' '
    return comment_words
a=words(a)

#Word Cloud
wc = WordCloud(max_font_size=30, max_words=100, background_color="white
wc.generate(a)
plt.figure(figsize=[15,10])
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```



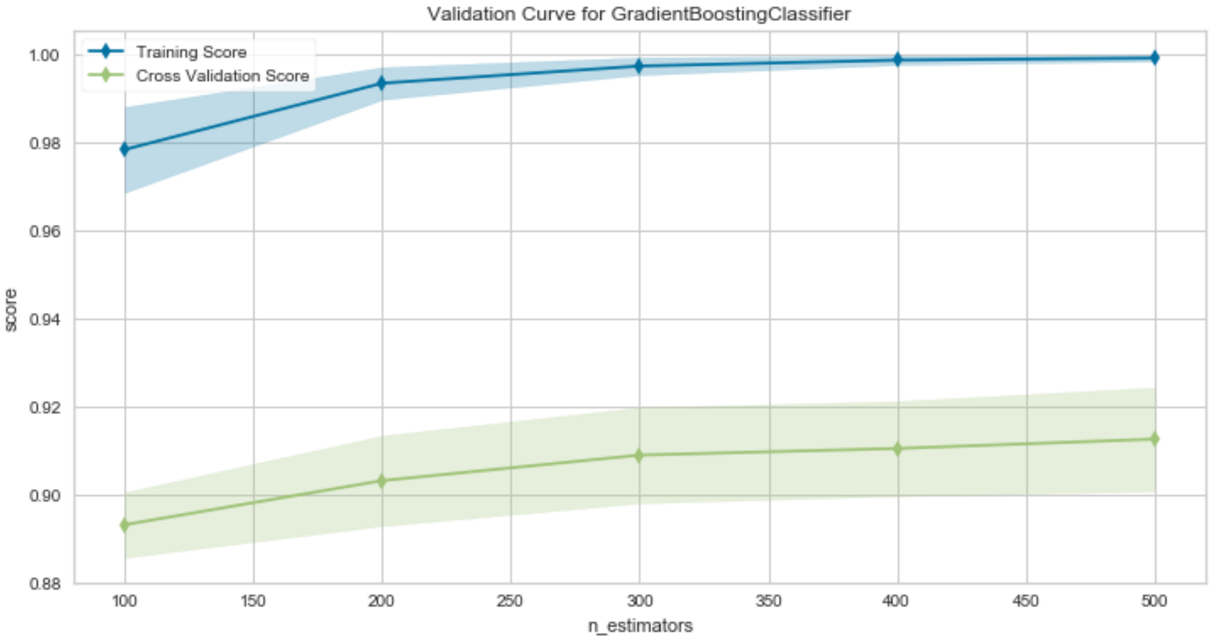
In [55]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from yellowbrick.model_selection import ValidationCurve

base_learners = [100,200,300,400,500]
depth = (range(1,20,4))
Learning_rate = [0.01,0.05,0.1,0.2,0.3]

param_grid = {'n_estimators': base_learners, 'max_depth':depth, 'learn
tscv = TimeSeriesSplit(n_splits=5)
gb = GradientBoostingClassifier(max_features='sqrt')
gsv = GridSearchCV(gb, param_grid,scoring='roc_auc',cv=tscv,n_jobs=-1,
gsv.fit(x_train, y_train)
cv_scores = gsv.cv_results_['mean_test_score']
print("Model with best parameters :\n",gsv.best_estimator_)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
optimal_n_estimators = gsv.best_estimator_.n_estimators
optimal_max_depth = gsv.best_estimator_.max_depth
optimal_learning_rate = gsv.best_estimator_.learning_rate
```

```
Fitting 5 folds for each of 125 candidates, totalling 625 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent w
orkers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed: 25.9min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed: 61.3min
[Parallel(n_jobs=-1)]: Done 625 out of 625 | elapsed: 89.2min finish
ed

Model with best parameters :
 GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=9,
              max_features='sqrt', max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=500,
              n_iter_no_change=None, presort='auto', random_state=No
ne,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
Best HyperParameter:  {'learning_rate': 0.1, 'max_depth': 9, 'n_esti
mators': 500}
Best Accuracy: 91.24%
```

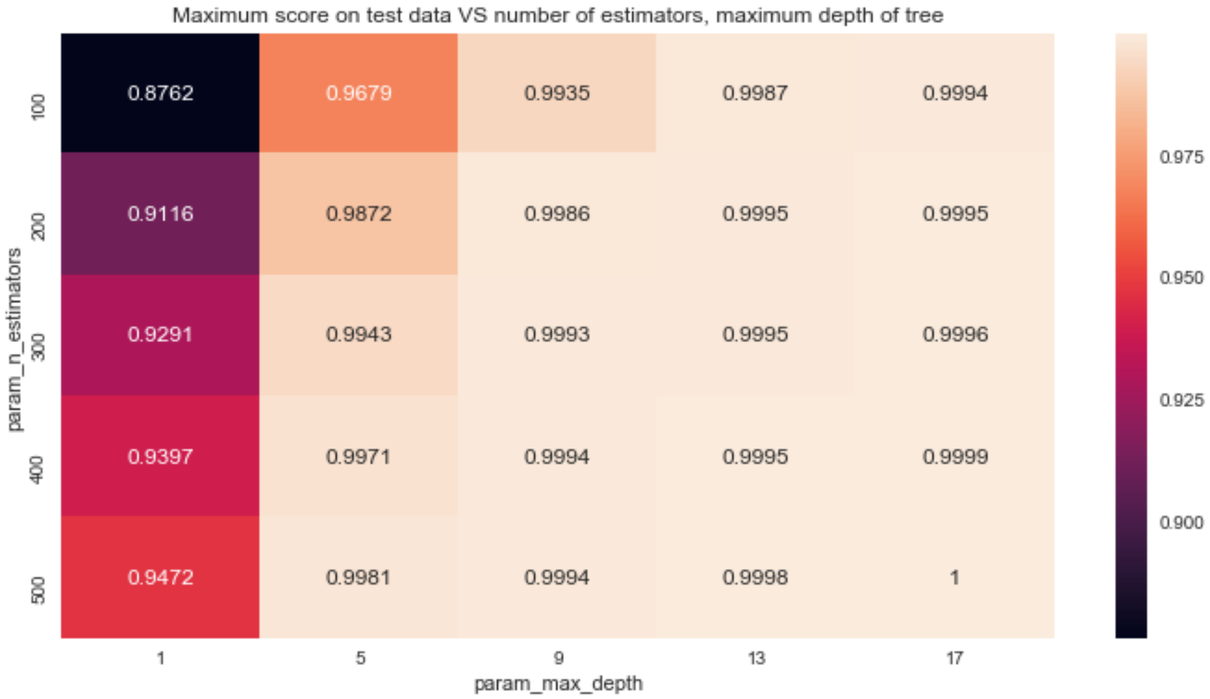
In [56]:

```
#Validation Curve
viz = ValidationCurve(gsv.best_estimator_, param_name="n_estimators",pa
viz.fit(x_train, y_train)
viz.poof()
```

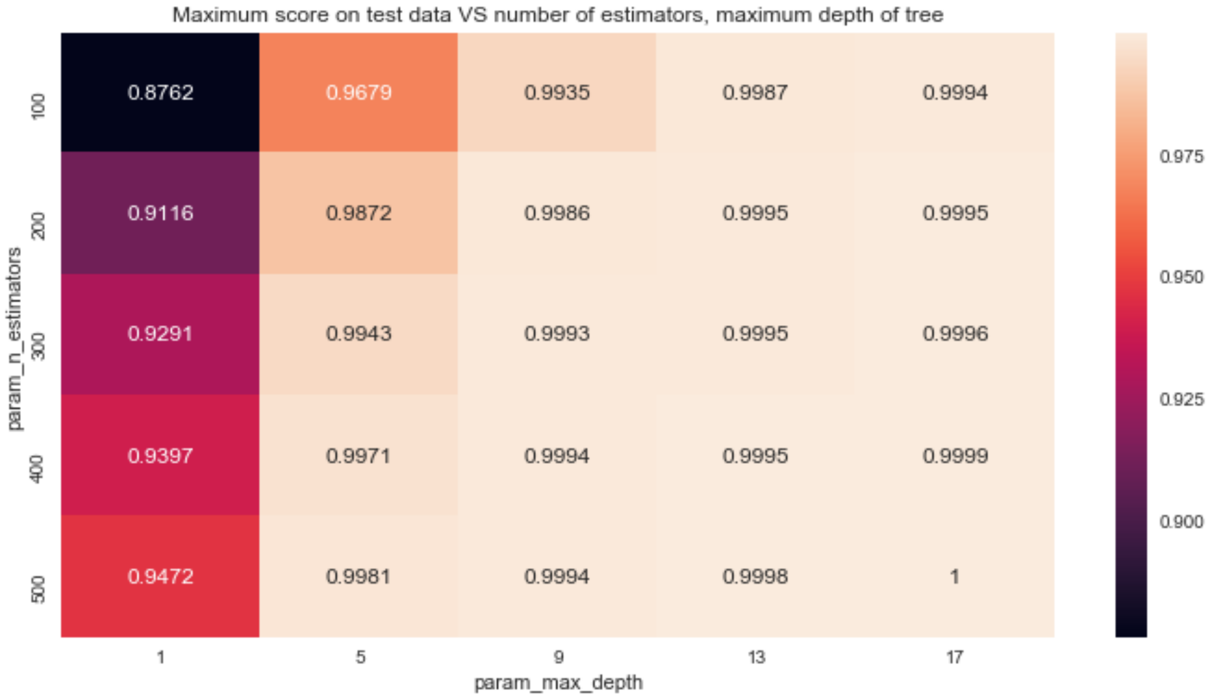


In [57]:

```
df_gridsearch = pd.DataFrame(gsv.cv_results_)
max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_de
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Maximum score on test data VS number of estimators, maximum de
fmt = 'png'
sns.heatmap(max_scores.mean_train_score, annot=True, fmt='.4g');
plt.title(title);
```



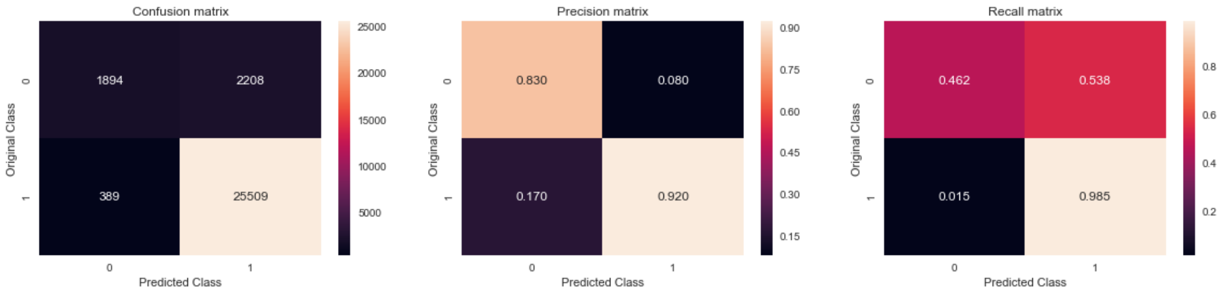
In [58]:
```
df_gridsearch = pd.DataFrame(gsv.cv_results_)
max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_de
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Maximum score on test data VS number of estimators, maximum de
fmt = 'png'
sns.heatmap(max_scores.mean_train_score, annot=True, fmt='.4g');
plt.title(title);
```



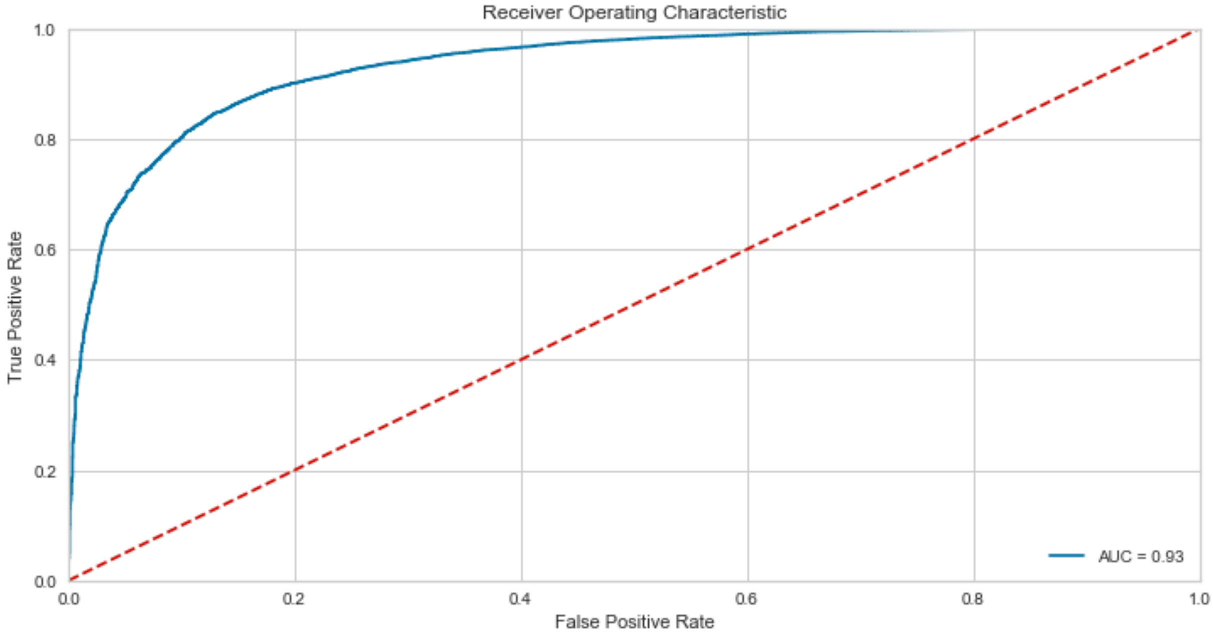
In [59]:

```
#Testing Accuracy on Test data
#Metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score



#Testing Accuracy on Test data
print("The optimal value of n_estimators is : ",optimal_n_estimators)
print("The optimal value of max_depth is : ",optimal_max_depth)

gb = GradientBoostingClassifier(n_estimators=optimal_n_estimators, max_
gb.fit(x_train,y_train)
y_pred = gb.predict(x_test)
y_prob = gb.predict_proba(x_test)


print("ROC_AUC on test set: %0.3f" % roc_auc_score(y_test, y_prob[:,1]
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
plot_confusion_matrix(y_test, y_pred)
```

```
The optimal value of n_estimators is :  500
The optimal value of max_depth is :  9
ROC_AUC on test set: 0.933
Accuracy on test set: 91.343%
Precision on test set: 0.920
Recall on test set: 0.985
F1-Score on test set: 0.952
```

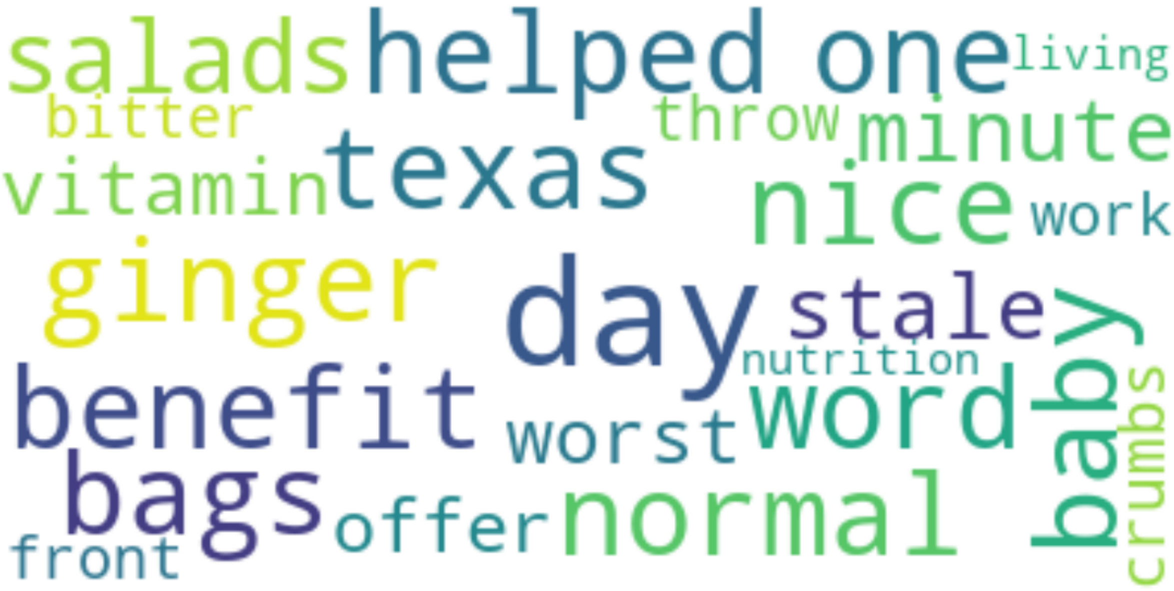


In [60]:
```
preds = y_prob[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
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 [61]:

```
#https://www.datacamp.com/community/tutorials/wordcloud-python
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt

importances=gb.feature_importances_
feat_names=count_vect.get_feature_names()

# Sort feature importances in descending order
indices = np.argsort(importances)[::-1][:25]
a=np.take(feat_names,indices)
def words(X):
    comment_words=' '
    for words in X:
        comment_words = comment_words + words + ' '
    return comment_words
a=words(a)

#Word Cloud
wc = WordCloud(max_font_size=50, max_words=100, background_color="white
wc.generate(a)
plt.figure(figsize=[20,10])
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```



[4.4] Word2Vec

In [62]:

```
# Random sampling
data = final.head(50000)

data_preprocessed_reviews=preprocessed_reviews[0:50000]
print("The size of sampled data is ",data.shape)
print("The size of sampled data is ",len(data_preprocessed_reviews))
```

```
The size of sampled data is  (50000, 10)
The size of sampled data is  50000
```

In [63]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data_preprocessed_
```

In [64]:

```
# Train your own Word2Vec model using your train text corpus
i=0
list_of_train_sentance=[]
for sentance in x_train:
    list_of_train_sentance.append(sentance.split())
```

In [65]:

```
# Train your own Word2Vec model using your test text corpus
i=0
list_of_test_sentance=[]
for sentance in x_test:
    list_of_test_sentance.append(sentance.split())
```

In [66]:
```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ra
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as va
# To use this code-snippet, download "GoogleNews-vectors-negative300.b
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/ed
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W1
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_train_sentance,min_count=5,size=50, wor
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vector
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_tra
```

```
[('wonderful', 0.8135449290275574), ('terrific', 0.812795102596283),
('excellent', 0.8054680824279785), ('fantastic', 0.804602861404419),
('awesome', 0.8004195690155029), ('good', 0.770828366279602), ('amaz
ing', 0.7531952857971191), ('perfect', 0.7331769466400146), ('fabulo
us', 0.7025632858276367), ('incredible', 0.6710256338119507)]
==================================================
[('greatest', 0.8081390857696533), ('best', 0.790654718875885), ('di
sgusting', 0.7568316459655762), ('tastiest', 0.7370363473892212), ('
closest', 0.7214698791503906), ('sweetest', 0.6875219345092773), ('n
icest', 0.6820852160453796), ('coolest', 0.6784370541572571), ('expe
rienced', 0.6771931648254395), ('awful', 0.6766822338104248)]
```

In [67]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  11561
sample words  ['witty', 'little', 'book', 'makes', 'son', 'laugh', '
loud', 'car', 'driving', 'along', 'always', 'sing', 'learned', 'indi
a', 'roses', 'love', 'new', 'words', 'classic', 'willing', 'bet', 's
till', 'able', 'memory', 'college', 'remember', 'seeing', 'show', 't
elevision', 'years', 'ago', 'child', 'sister', 'later', 'bought', 'd
ay', 'thirty', 'something', 'used', 'series', 'books', 'songs', 'stu
dent', 'teaching', 'turned', 'whole', 'school', 'purchasing', 'cd',
'children']
```

[4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [68]:
```
average Word2Vec
compute average word2vec for each review in Train data.
nt_train_vectors = []; # the avg-w2v for each sentence/review is stored
r sent in tqdm(list_of_train_sentance): # for each review/sentence
  sent_vec = np.zeros(50) # as word vectors are of zero length 50, you
  cnt_words =0; # num of words with a valid vector in the sentence/revi
  for word in sent: # for each word in a review/sentence
      if word in w2v_words:
          vec = w2v_model.wv[word]
          sent_vec += vec
          cnt_words += 1
  if cnt_words != 0:
      sent_vec /= cnt_words
  sent_train_vectors.append(sent_vec)
int(len(sent_train_vectors))
int(len(sent_train_vectors[0]))
```

```
100%|██████████| 35000/35000 [01:27<00:00, 399.56it/s]

35000
50
```

In [69]:

```
# average Word2Vec
# compute average word2vec for each review in Test data.
sent_test_vectors = []; # the avg-w2v for each sentence/review is stor
for sent in tqdm(list_of_test_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, y
    cnt_words =0; # num of words with a valid vector in the sentence/r
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_test_vectors.append(sent_vec)
print(len(sent_test_vectors))
print(len(sent_test_vectors[0]))
```

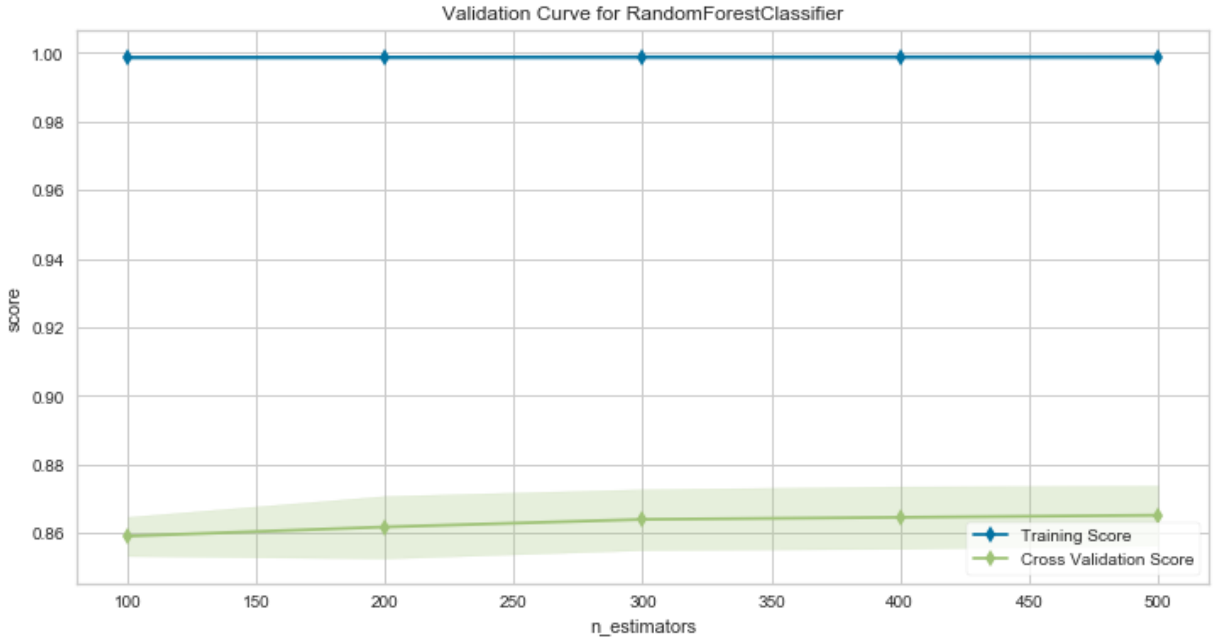
```
100%|██████████| 15000/15000 [00:39<00:00, 380.74it/s]

15000
50
```

In [70]:

```
rom sklearn.ensemble import RandomForestClassifier
rom sklearn.model_selection import TimeSeriesSplit
rom sklearn.model_selection import GridSearchCV
rom yellowbrick.model_selection import ValidationCurve

ase_learners = [100,200,300,400,500]
epth = (range(1,20,4))
aram_grid={"n_estimators":base_learners,"max_depth":depth}
scv = TimeSeriesSplit(n_splits=5)
f = RandomForestClassifier(max_features='sqrt', class_weight="balanced")
sv = GridSearchCV(rf, param_grid,scoring='roc_auc',cv=tscv,n_jobs=-1,ve
sv.fit(sent_train_vectors, y_train)
_scores = gsv.cv_results_['mean_test_score']
rint("Model with best parameters :\n",gsv.best_estimator_)
rint("Best Score: %.2f%%"%(gsv.best_score_*100))
rint("Best HyperParameter: ",gsv.best_params_)
ptimal_n_estimators = gsv.best_estimator_.n_estimators
ptimal_max_depth = gsv.best_estimator_.max_depth

Validation Curve
iz = ValidationCurve(gsv.best_estimator_, param_name="n_estimators",para
iz.fit(sent_train_vectors, y_train)
iz.poof()
```

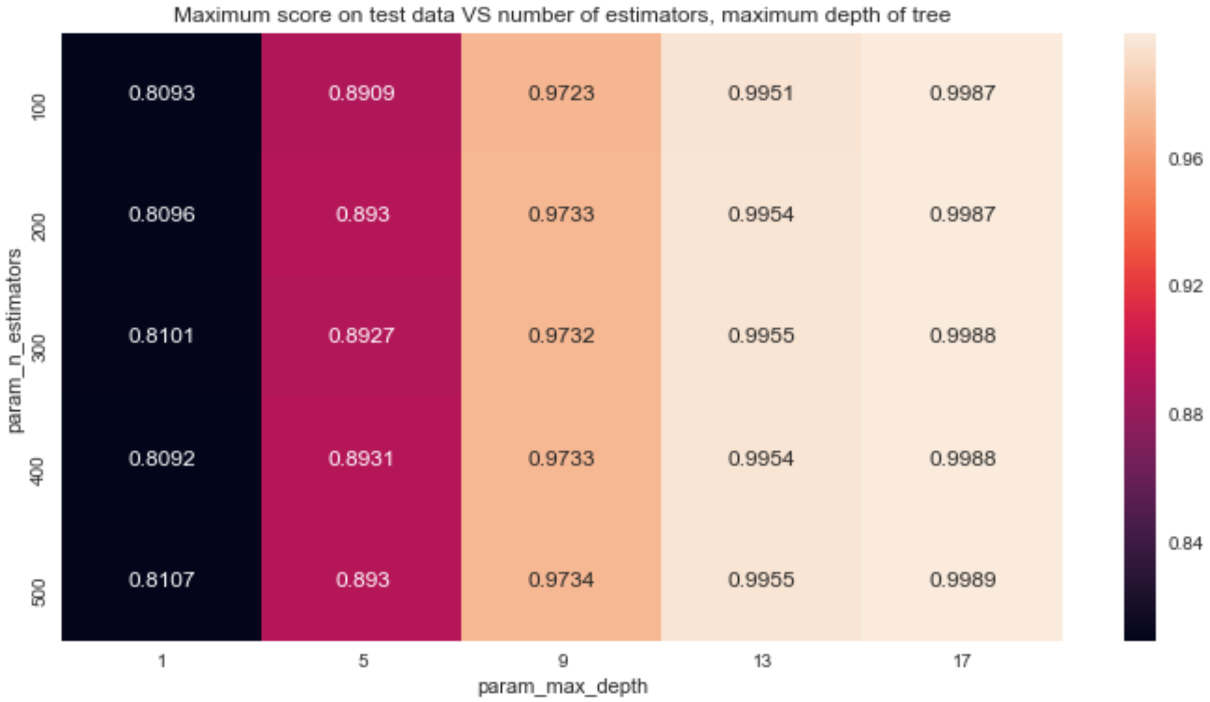
```
Fitting 5 folds for each of 25 candidates, totalling 125 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent w
orkers.
```

```
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  3.0min
[Parallel(n_jobs=-1)]: Done 125 out of 125 | elapsed: 19.2min finish
ed

Model with best parameters :
 RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=17, max_features='sqrt',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=500, n_jobs=None, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
Best Score: 86.45%
Best HyperParameter:  {'max_depth': 17, 'n_estimators': 500}
```

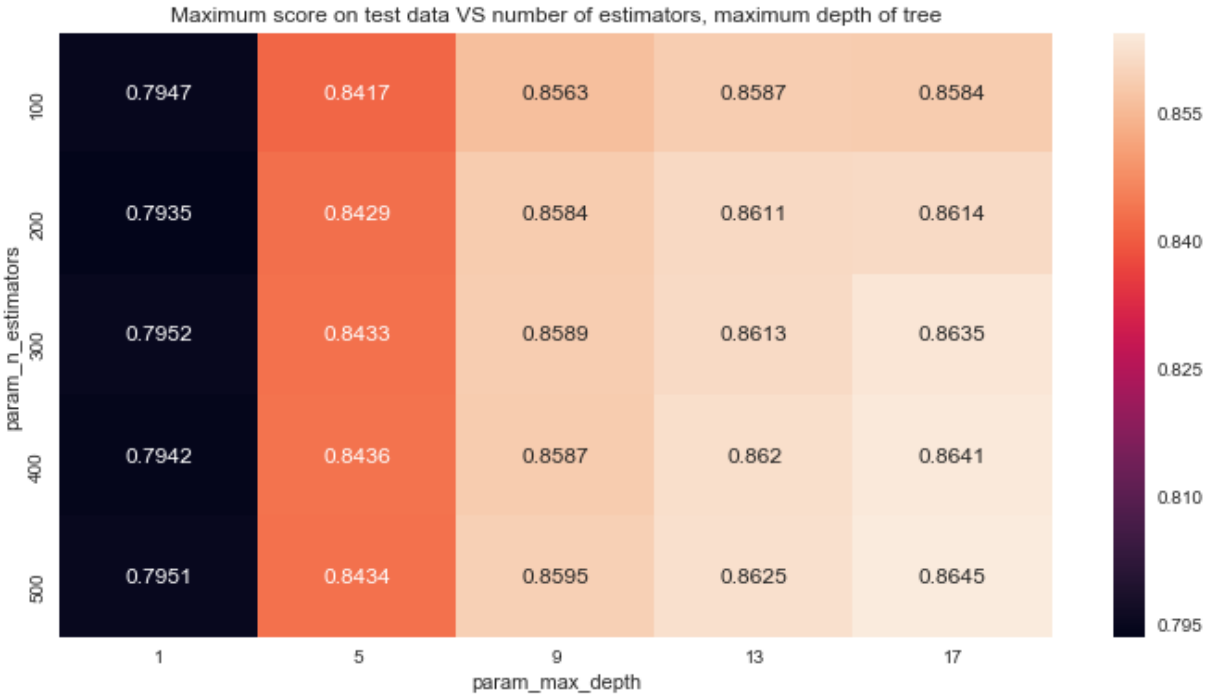


In [71]:

```
df_gridsearch = pd.DataFrame(gsv.cv_results_)
max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_de
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Maximum score on test data VS number of estimators, maximum de
fmt = 'png'
sns.heatmap(max_scores.mean_train_score, annot=True, fmt='.4g');
plt.title(title);
```

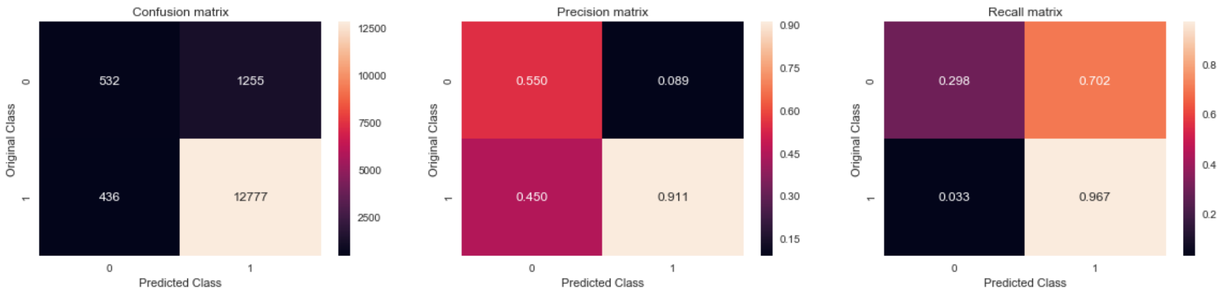


In [72]:
```
df_gridsearch = pd.DataFrame(gsv.cv_results_)
max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_de
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Maximum score on test data VS number of estimators, maximum de
fmt = 'png'
sns.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g');
plt.title(title);
```



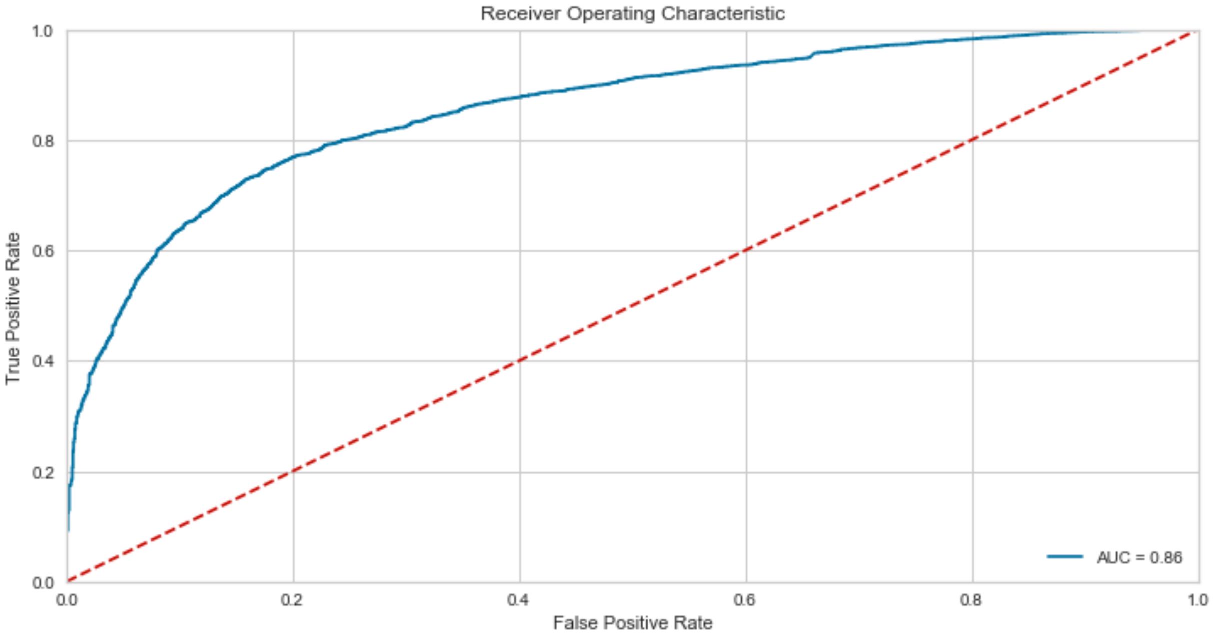
```
In [73]: esting Accuracy on Test data
         etrics
         om sklearn.metrics import accuracy_score
         om sklearn.metrics import confusion_matrix
         om sklearn.metrics import precision_score
         om sklearn.metrics import recall_score
         om sklearn.metrics import f1_score
         om sklearn.metrics import roc_curve, auc
         om sklearn.metrics import roc_auc_score



         esting Accuracy on Test data
         int("The optimal value of n_estimators is : ",optimal_n_estimators)
         int("The optimal value of max_depth is : ",optimal_max_depth)
          = RandomForestClassifier(n_estimators=optimal_n_estimators, max_depth=
         .fit(sent_train_vectors,y_train)
         pred = rf.predict(sent_test_vectors)
         prob = rf.predict_proba(sent_test_vectors)


         int("ROC_AUC on test set: %0.3f" % roc_auc_score(y_test, y_prob[:,1]))
         int("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100
         int("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
         int("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
         int("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
         ot_confusion_matrix(y_test, y_pred)
```

```
The optimal value of n_estimators is :  500
The optimal value of max_depth is :  17
ROC_AUC on test set: 0.857
Accuracy on test set: 88.727%
Precision on test set: 0.911
Recall on test set: 0.967
F1-Score on test set: 0.938
```



In [74]:

```
preds = y_prob[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
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()
```

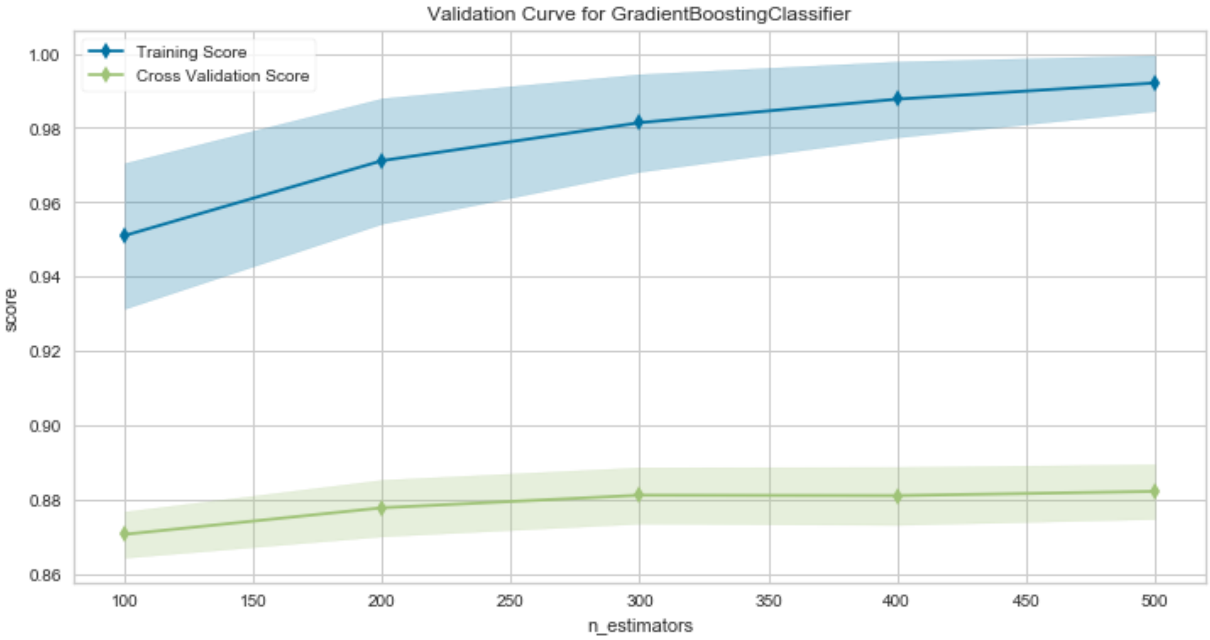


In [75]:

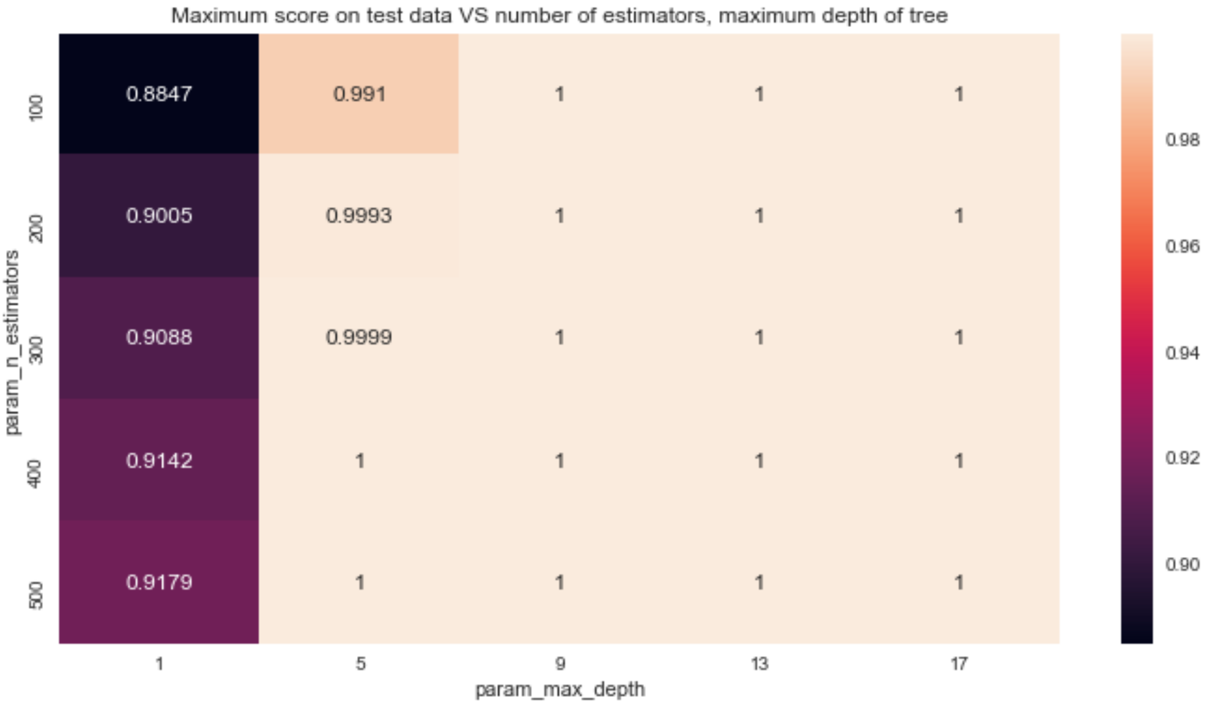
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit

from yellowbrick.model_selection import ValidationCurve

base_learners = [100,200,300,400,500]
depth = (range(1,20,4))
Learning_rate = [0.01,0.05,0.1,0.2,0.3]

param_grid = {'n_estimators': base_learners, 'max_depth':depth, 'learn
tscv = TimeSeriesSplit(n_splits=5)
gb = GradientBoostingClassifier(max_features='sqrt')
gsv = GridSearchCV(gb, param_grid,scoring='roc_auc',cv=tscv,n_jobs=-1,
gsv.fit(sent_train_vectors, y_train)
cv_scores = gsv.cv_results_['mean_test_score']
print("Model with best parameters :\n",gsv.best_estimator_)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
optimal_n_estimators = gsv.best_estimator_.n_estimators
optimal_max_depth = gsv.best_estimator_.max_depth
optimal_learning_rate = gsv.best_estimator_.learning_rate
```

```
Fitting 5 folds for each of 125 candidates, totalling 625 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent w
orkers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  3.4min
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed: 129.8min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed: 241.8min
[Parallel(n_jobs=-1)]: Done 625 out of 625 | elapsed: 280.9min finis
hed

Model with best parameters :
 GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.05, loss='deviance', max_depth=5,
              max_features='sqrt', max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=500,
              n_iter_no_change=None, presort='auto', random_state=No
ne,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
Best HyperParameter:  {'learning_rate': 0.05, 'max_depth': 5, 'n_est
imators': 500}
Best Accuracy: 88.28%
```

In [76]:
```
#Validation Curve
viz = ValidationCurve(gsv.best_estimator_, param_name="n_estimators",pa
viz.fit(sent_train_vectors, y_train)
viz.poof()
```

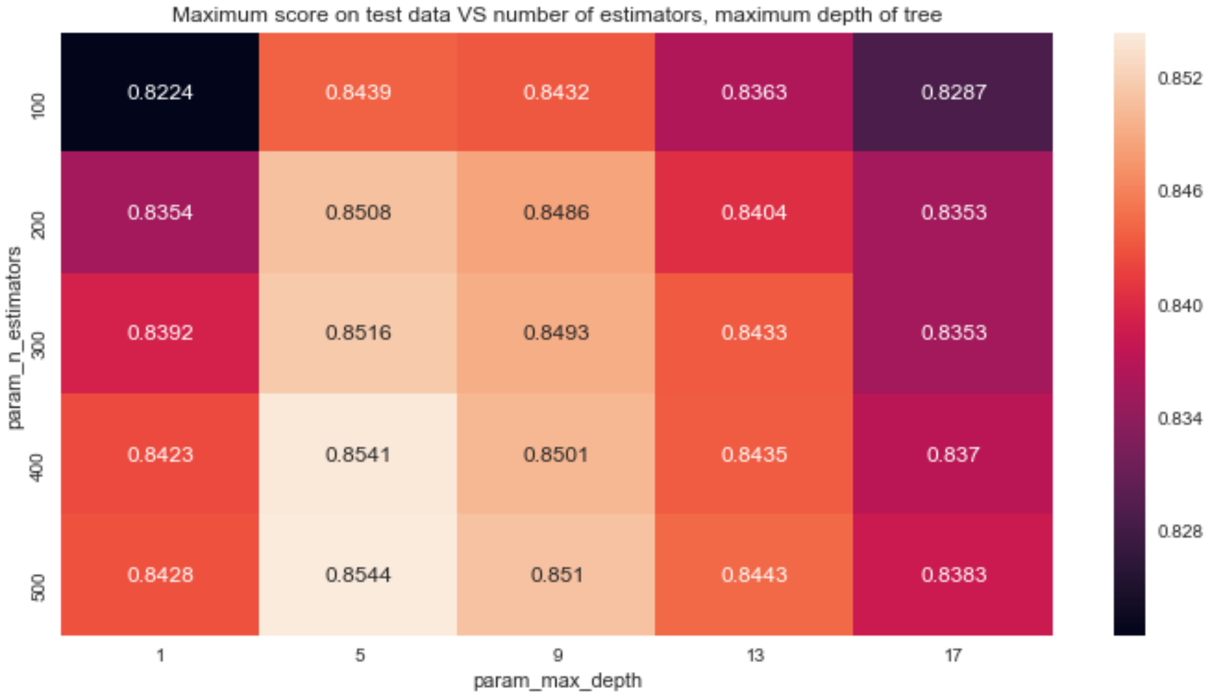


In [78]:
```
df_gridsearch = pd.DataFrame(gsv.cv_results_)
max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_dep
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Maximum score on test data VS number of estimators, maximum de
fmt = 'png'
sns.heatmap(max_scores.mean_train_score, annot=True, fmt='.4g');
plt.title(title);
```



In [95]:

```
df_gridsearch = pd.DataFrame(gsv.cv_results_)
max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_de
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Maximum score on test data VS number of estimators, maximum d
fmt = 'png'
sns.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g');
plt.title(title);
```

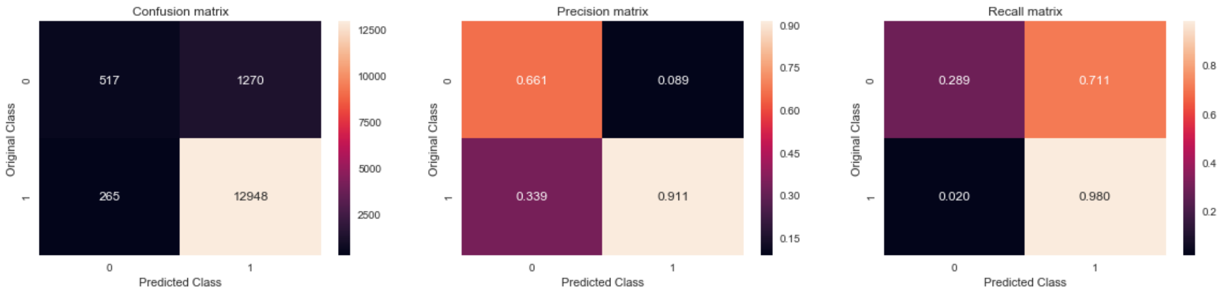


	1	5	9	13	17
100	0.8224	0.8439	0.8432	0.8363	0.8287
200	0.8354	0.8508	0.8486	0.8404	0.8353
300	0.8392	0.8516	0.8493	0.8433	0.8353
400	0.8423	0.8541	0.8501	0.8435	0.837
500	0.8428	0.8544	0.851	0.8443	0.8383

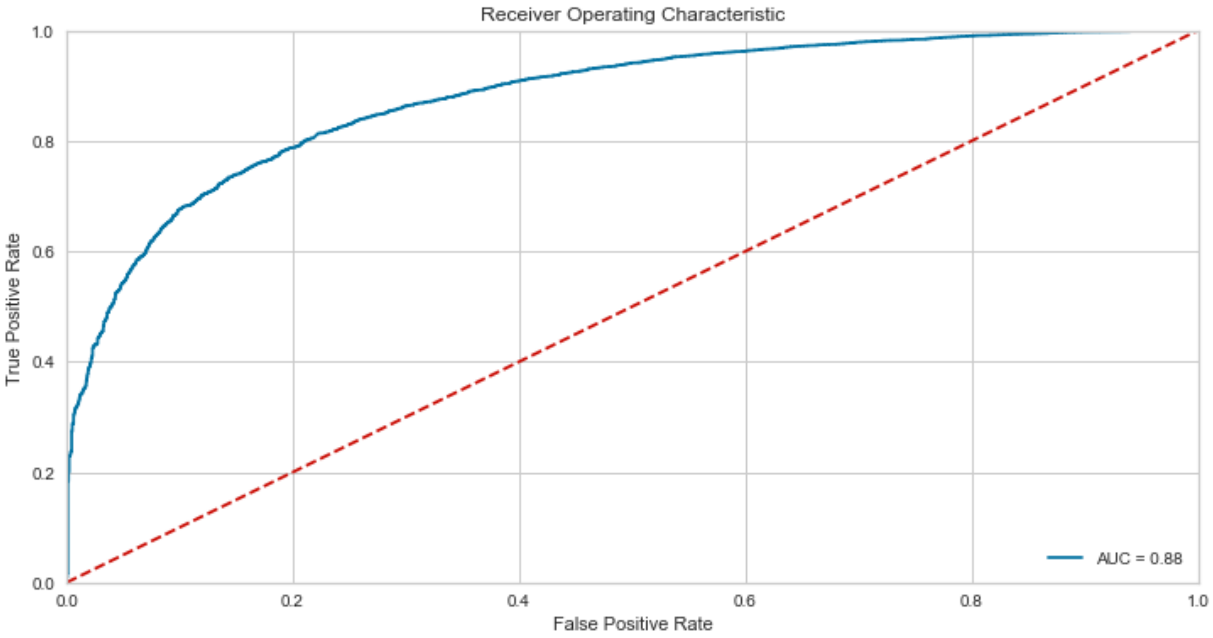
In [79]:

```
sting Accuracy on Test data
trics
m sklearn.metrics import accuracy_score
m sklearn.metrics import confusion_matrix
m sklearn.metrics import precision_score
m sklearn.metrics import recall_score
m sklearn.metrics import f1_score
m sklearn.metrics import roc_curve, auc
m sklearn.metrics import roc_auc_score



sting Accuracy on Test data
nt("The optimal value of n_estimators is : ",optimal_n_estimators)
nt("The optimal value of max_depth is : ",optimal_max_depth)

= GradientBoostingClassifier(n_estimators=optimal_n_estimators, max_dep
fit(sent_train_vectors,y_train)
red = gb.predict(sent_test_vectors)
rob = gb.predict_proba(sent_test_vectors)


nt("ROC_AUC on test set: %0.3f" % roc_auc_score(y_test, y_prob[:,1]))
nt("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100)
nt("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
nt("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
nt("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
t_confusion_matrix(y_test, y_pred)
```

```
The optimal value of n_estimators is :  500
The optimal value of max_depth is :  5
ROC_AUC on test set: 0.878
Accuracy on test set: 89.767%
Precision on test set: 0.911
Recall on test set: 0.980
F1-Score on test set: 0.944
```



In [80]:
```
preds = y_prob[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
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()
```



[4.4.1.2] TFIDF weighted W2v

In [81]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer(ngram_range=(1,1))
tf_idf_matrix = model.fit_transform(data_preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [82]:

```
# TF-IDF weighted Word2Vec on Train data
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and c

train_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/revie
row=0;
for sent in tqdm(list_of_train_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    train_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 35000/35000 [18:34<00:00, 31.39it/s]
```

In [83]:

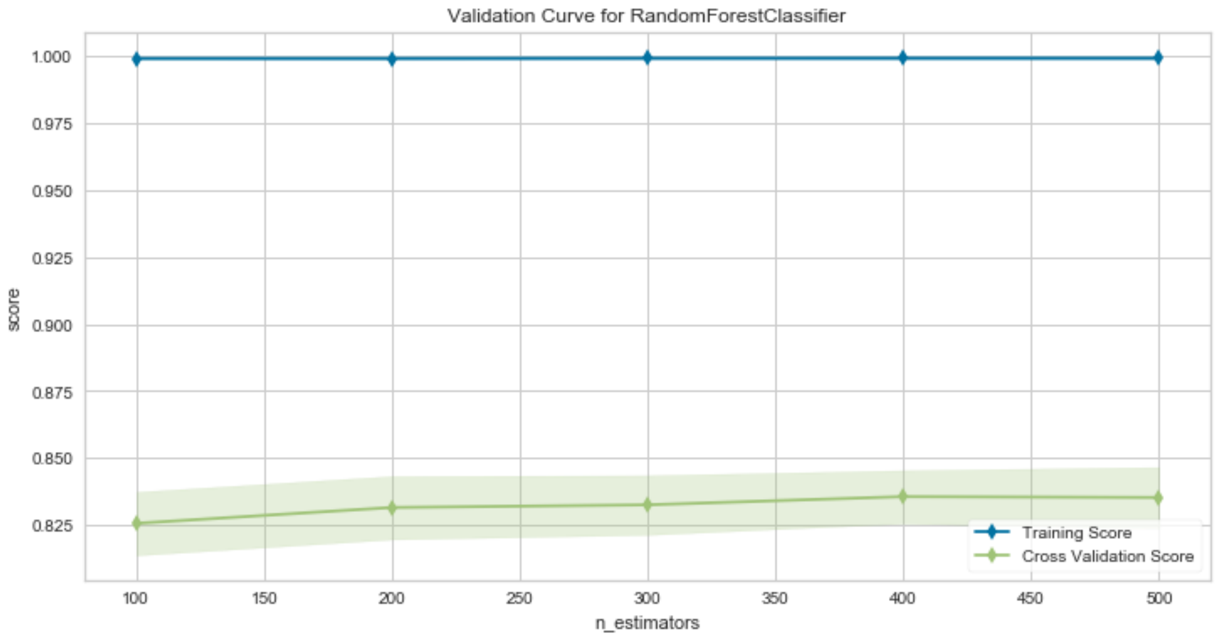
```
# TF-IDF weighted Word2Vec on Test data

test_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review
row=0;
for sent in tqdm(list_of_test_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    test_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 15000/15000 [08:18<00:00, 30.12it/s]
```

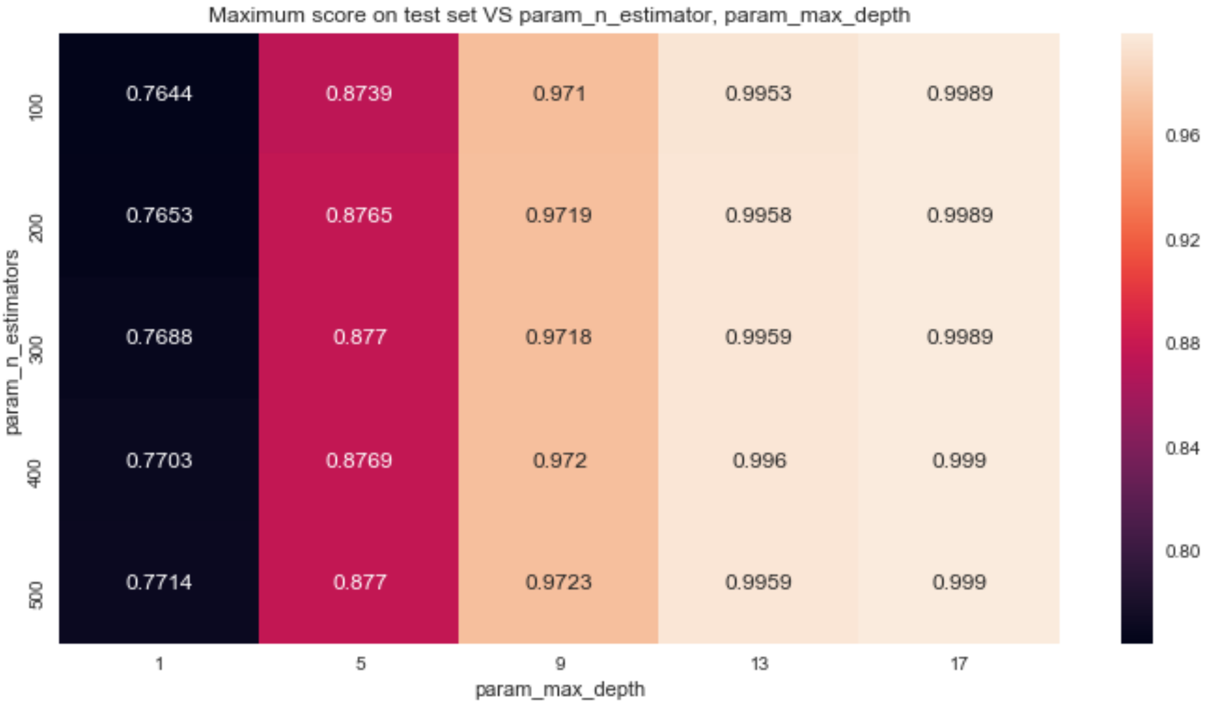
In [84]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import GridSearchCV
from yellowbrick.model_selection import ValidationCurve

base_learners = [100,200,300,400,500]
depth = (range(1,20,4))
param_grid={"n_estimators":base_learners,"max_depth":depth}
tscv = TimeSeriesSplit(n_splits=5)
rf = RandomForestClassifier(max_features='sqrt', class_weight="balance
gsv = GridSearchCV(rf, param_grid,scoring='roc_auc',cv=tscv,n_jobs=-1,
gsv = GridSearchCV(rf, param_grid,scoring='roc_auc',cv=tscv,n_jobs=-1)
gsv.fit(train_tfidf_sent_vectors, y_train)
cv_scores = gsv.cv_results_['mean_test_score']
print("Model with best parameters :\n",gsv.best_estimator_)
print("Best Score: %.2f%%"%(gsv.best_score_*100))
print("Best HyperParameter: ",gsv.best_params_)
optimal_n_estimators = gsv.best_estimator_.n_estimators
optimal_max_depth = gsv.best_estimator_.max_depth

#Validation Curve
viz = ValidationCurve(gsv.best_estimator_, param_name="n_estimators",pa
viz.fit(train_tfidf_sent_vectors, y_train)
viz.poof()
```

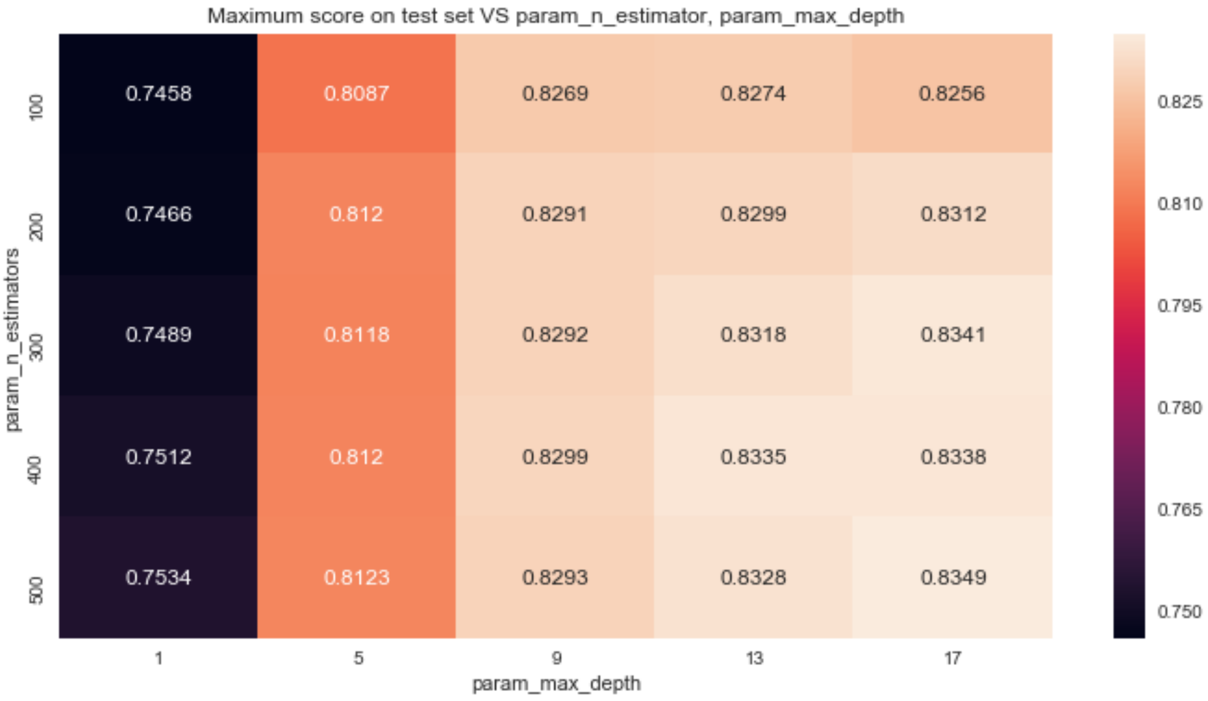
```
Fitting 5 folds for each of 25 candidates, totalling 125 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent w
orkers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  3.0min
/Users/rohitbohra/anaconda3/lib/python3.6/site-packages/sklearn/exte
rnals/joblib/externals/loky/process_executor.py:706: UserWarning: A
worker stopped while some jobs were given to the executor. This can
be caused by a too short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done 125 out of 125 | elapsed: 19.4min finish
ed

Model with best parameters :
 RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=17, max_features='sqrt',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=500, n_jobs=None, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
Best Score: 83.49%
Best HyperParameter:  {'max_depth': 17, 'n_estimators': 500}
```



In [85]:
```
df_gridsearch = pd.DataFrame(gsv.cv_results_)
max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_de
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Maximum score on test set VS param_n_estimator, param_max_dep
fmt = 'png'
sns.heatmap(max_scores.mean_train_score, annot=True, fmt='.4g');
plt.title(title);
```

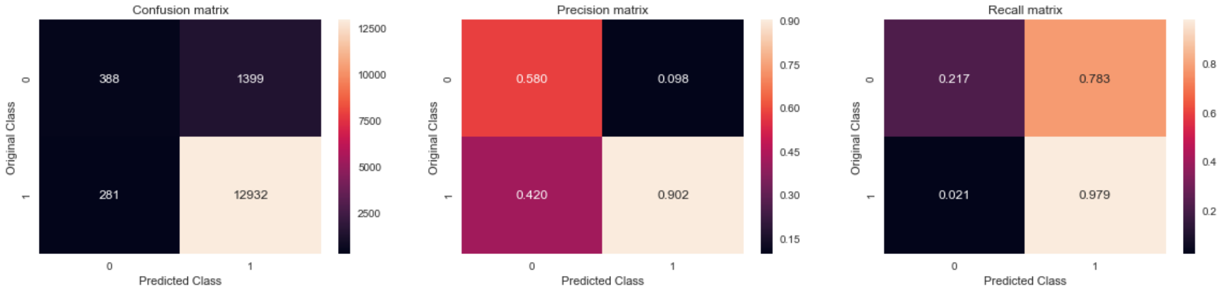


In [86]:
```
df_gridsearch = pd.DataFrame(gsv.cv_results_)
max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_de
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Maximum score on test set VS param_n_estimator, param_max_dep
fmt = 'png'
sns.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g');
plt.title(title);
```



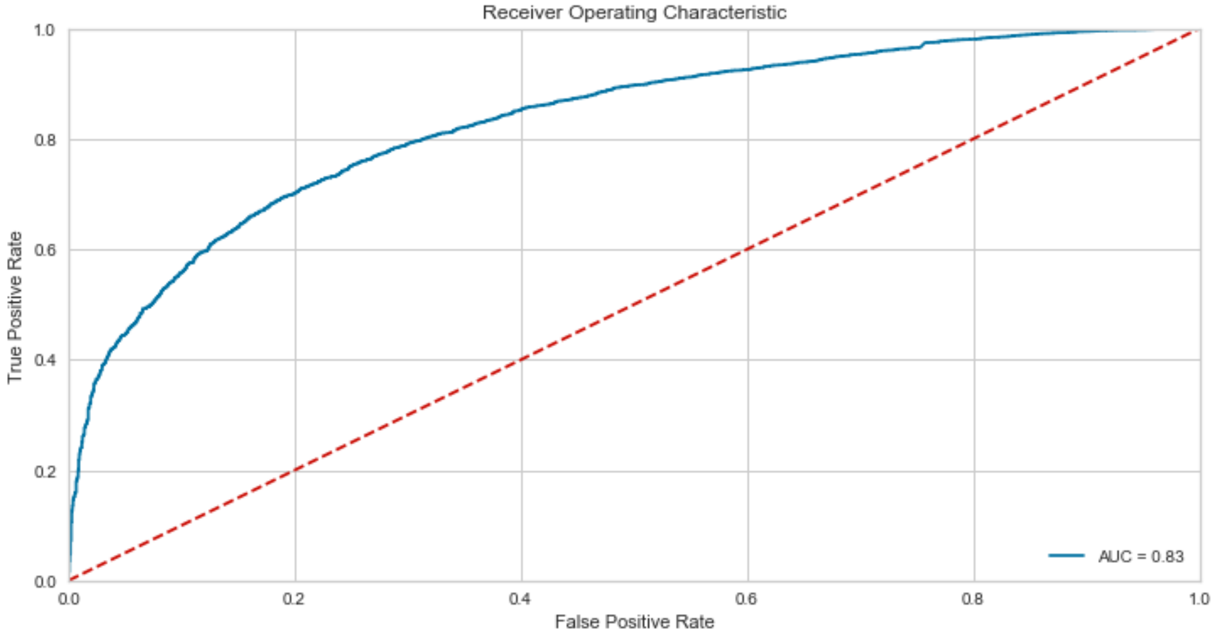
In [87]:
```
Testing Accuracy on Test data
Metrics
rom sklearn.metrics import accuracy_score
rom sklearn.metrics import confusion_matrix
rom sklearn.metrics import precision_score
rom sklearn.metrics import recall_score
rom sklearn.metrics import f1_score
rom sklearn.metrics import roc_curve, auc
rom sklearn.metrics import roc_auc_score



Testing Accuracy on Test data
rint("The optimal value of n_estimators is : ",optimal_n_estimators)
rint("The optimal value of max_depth is : ",optimal_max_depth)
f = RandomForestClassifier(n_estimators=optimal_n_estimators, max_depth
f.fit(sent_train_vectors,y_train)
_pred = rf.predict(test_tfidf_sent_vectors)
_prob = rf.predict_proba(test_tfidf_sent_vectors)


rint("ROC_AUC on test set: %0.3f" % roc_auc_score(y_test, y_prob[:,1]))
rint("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*10
rint("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
rint("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
rint("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
lot_confusion_matrix(y_test, y_pred)
```

```
The optimal value of n_estimators is :  500
The optimal value of max_depth is :  17
ROC_AUC on test set: 0.830
Accuracy on test set: 88.800%
Precision on test set: 0.902
Recall on test set: 0.979
F1-Score on test set: 0.939
```



In [88]:

```
preds = y_prob[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
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()
```



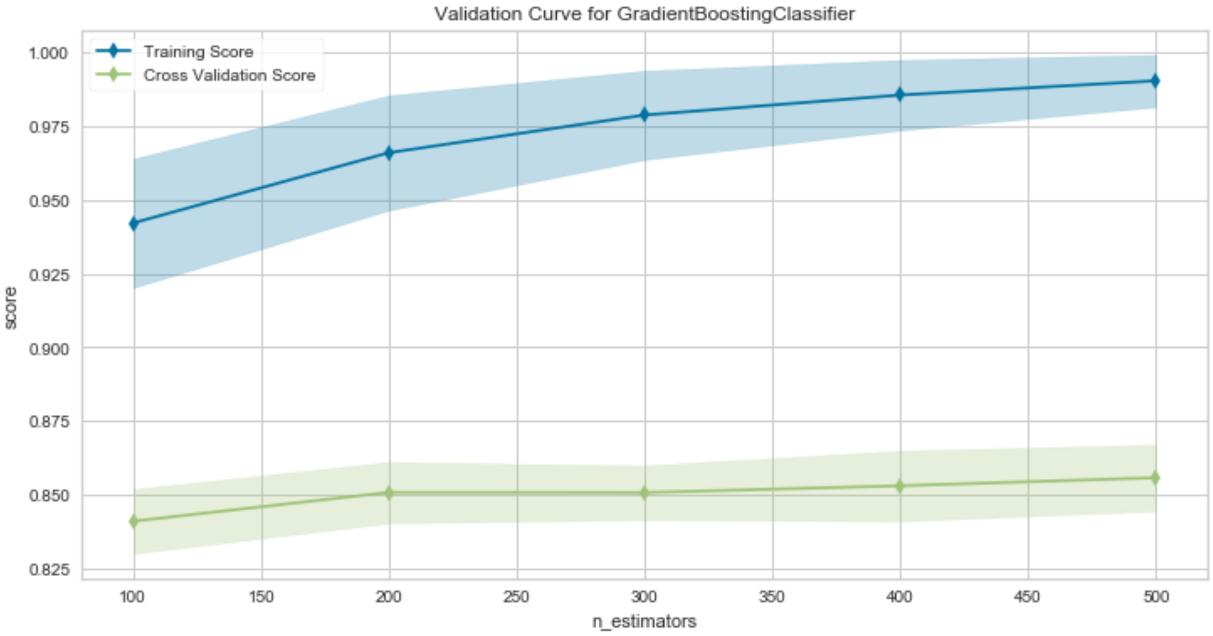
In [89]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit

from yellowbrick.model_selection import ValidationCurve

base_learners = [100,200,300,400,500]
depth = (range(1,20,4))
Learning_rate = [0.01,0.05,0.1,0.2,0.3]

param_grid = {'n_estimators': base_learners, 'max_depth':depth, 'learn
tscv = TimeSeriesSplit(n_splits=5)
gb = GradientBoostingClassifier(max_features='sqrt')
gsv = GridSearchCV(gb, param_grid,scoring='roc_auc',cv=tscv,n_jobs=-1,
gsv.fit(train_tfidf_sent_vectors, y_train)
cv_scores = gsv.cv_results_['mean_test_score']
print("Model with best parameters :\n",gsv.best_estimator_)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
optimal_n_estimators = gsv.best_estimator_.n_estimators
optimal_max_depth = gsv.best_estimator_.max_depth
optimal_learning_rate = gsv.best_estimator_.learning_rate
```

```
Fitting 5 folds for each of 125 candidates, totalling 625 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent w
orkers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed: 129.8min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed: 238.8min
[Parallel(n_jobs=-1)]: Done 625 out of 625 | elapsed: 284.0min finis
hed

Model with best parameters :
 GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.05, loss='deviance', max_depth=5,
              max_features='sqrt', max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=500,
              n_iter_no_change=None, presort='auto', random_state=No
ne,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
Best HyperParameter:  {'learning_rate': 0.05, 'max_depth': 5, 'n_est
imators': 500}
Best Accuracy: 85.44%
```

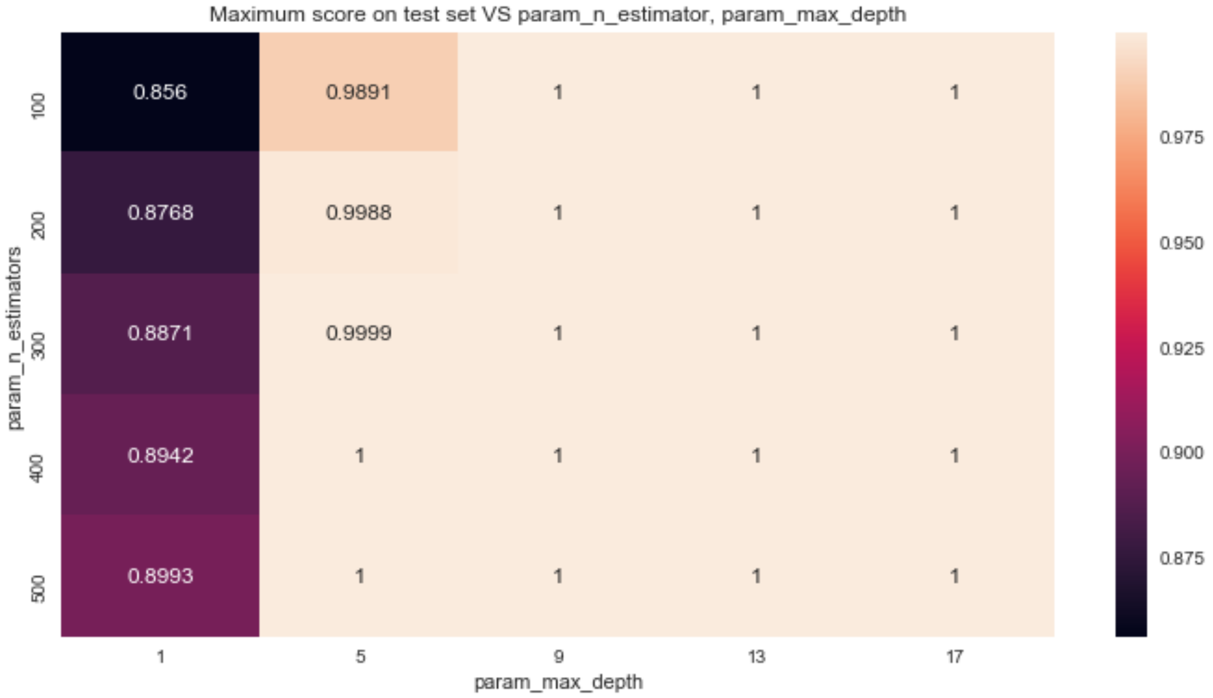
In [90]:

```
#Validation Curve
viz = ValidationCurve(gsv.best_estimator_, param_name="n_estimators",pa
viz.fit(train_tfidf_sent_vectors, y_train)
viz.poof()
```



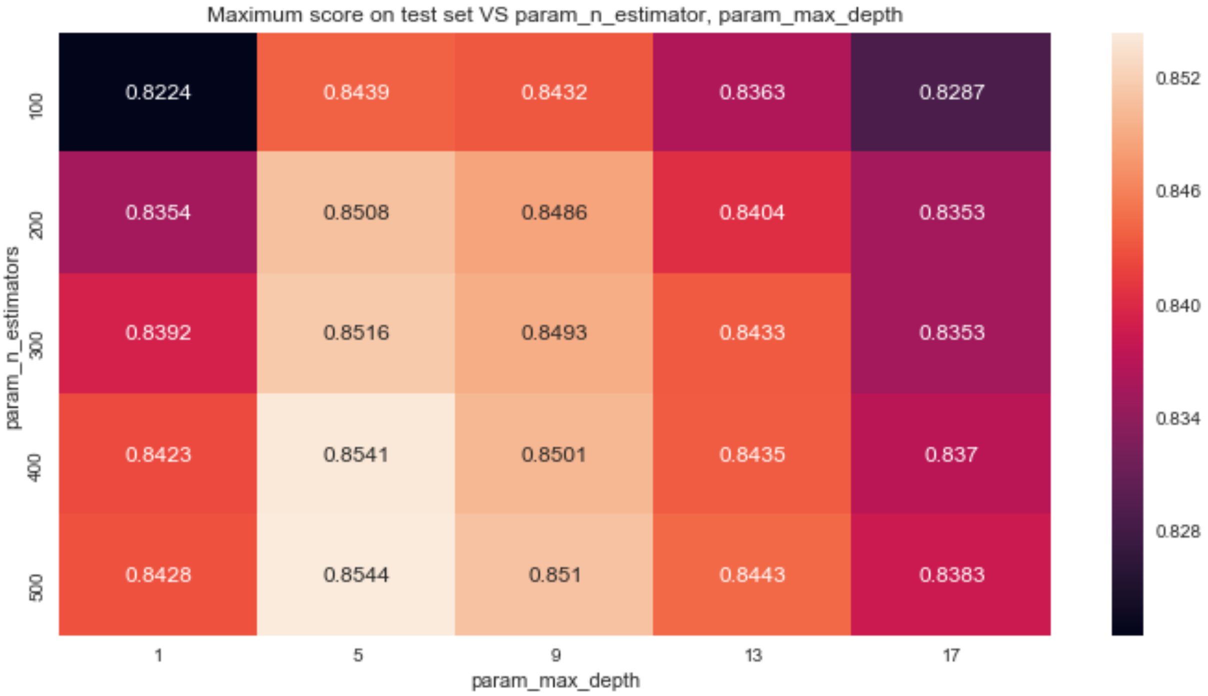
In [92]:

```
df_gridsearch = pd.DataFrame(gsv.cv_results_)
max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_de
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Maximum score on test set VS param_n_estimator, param_max_dep
fmt = 'png'
sns.heatmap(max_scores.mean_train_score, annot=True, fmt='.4g');
plt.title(title);
```



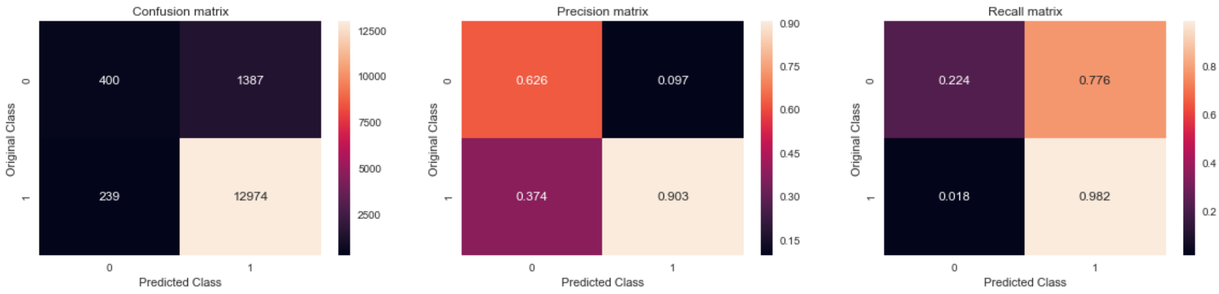
In [93]:

```
df_gridsearch = pd.DataFrame(gsv.cv_results_)
max_scores = df_gridsearch.groupby(['param_n_estimators','param_max_de
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Maximum score on test set VS param_n_estimator, param_max_dep
fmt = 'png'
sns.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g');
plt.title(title);
```



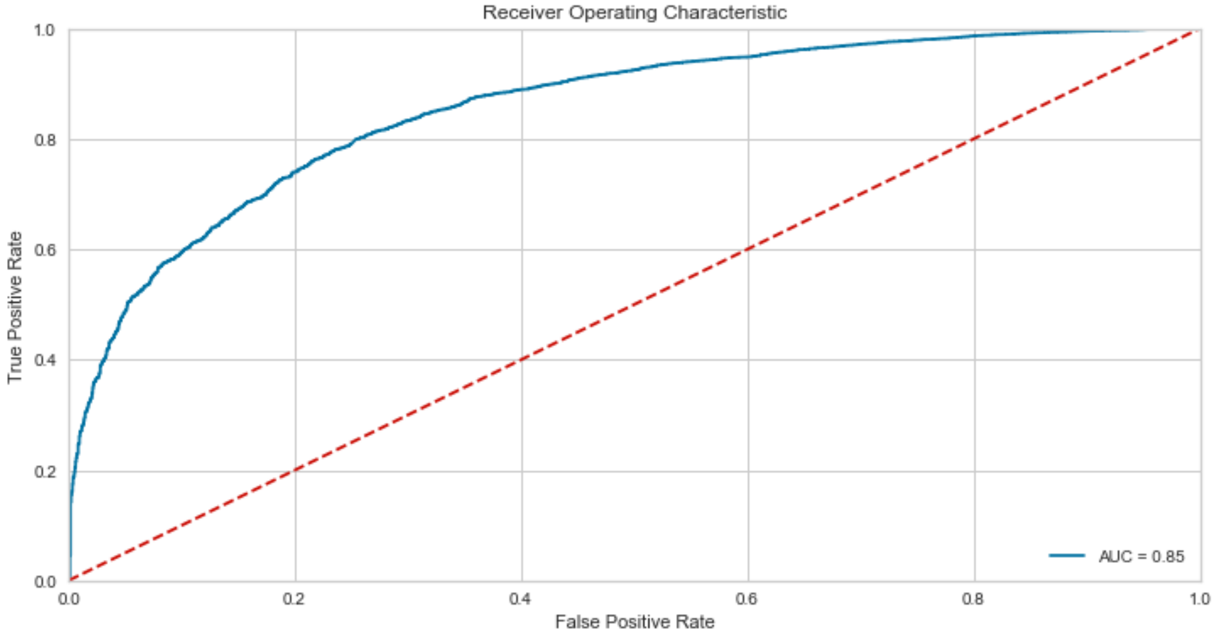
```
In [91]: Testing Accuracy on Test data
         Metrics
         rom sklearn.metrics import accuracy_score
         rom sklearn.metrics import confusion_matrix
         rom sklearn.metrics import precision_score
         rom sklearn.metrics import recall_score
         rom sklearn.metrics import f1_score
         rom sklearn.metrics import roc_curve, auc
         rom sklearn.metrics import roc_auc_score



         Testing Accuracy on Test data
         rint("The optimal value of n_estimators is : ",optimal_n_estimators)
         rint("The optimal value of max_depth is : ",optimal_max_depth)

         o = GradientBoostingClassifier(n_estimators=optimal_n_estimators, max_de
         o.fit(train_tfidf_sent_vectors,y_train)
         _pred = gb.predict(test_tfidf_sent_vectors)
         _prob = gb.predict_proba(test_tfidf_sent_vectors)


         rint("ROC_AUC on test set: %0.3f" % roc_auc_score(y_test, y_prob[:,1]))
         rint("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*10
         rint("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
         rint("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
         rint("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
         lot_confusion_matrix(y_test, y_pred)
```

```
The optimal value of n_estimators is :  500
The optimal value of max_depth is :  5
ROC_AUC on test set: 0.855
Accuracy on test set: 89.160%
Precision on test set: 0.903
Recall on test set: 0.982
F1-Score on test set: 0.941
```



In [94]:

```
preds = y_prob[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
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 [98]:

```
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ["Ensemble Technique","Featurization", "Number of Esti

x.add_row(["Random Forest","BOW", 500,13,0.898, 0.900, 83.733])
x.add_row(["Gradient Boosting DT","BOW",400,9,0.932, 0.951, 91.210])
x.add_row(["Random Forest","TF-IDF",500,17,0.904, 0.914, 85.827])
x.add_row(["Gradient Boosting DT","TF-IDF", 500,9,0.933, 0.952, 91.343
x.add_row(["Random Forest","Avg. W2V", 500,17,0.857, 0.938, 88.727])
x.add_row(["Gradient Boosting DT","Avg-W2V", 500,5,0.878, 0.944, 89.76
x.add_row(["Random Forest","TF-IDF Avg. W2V", 500,17,0.830, 0.939, 88.
x.add_row(["Gradient Boosting DT","TF-IDF Avg. W2V", 500,5,0.855, 0.94
print('\t\t\t\t\t\tEnsemble Modeling')
print(x)
```

```
                                                Ensemble Modeling
+----------------------+-----------------+----------------------+----
-----------+---------+----------+----------+
|  Ensemble Technique  |  Featurization  | Number of Estimator | Max
imum Depth | ROC-AUC | F1-Score | Accuracy |
+----------------------+-----------------+----------------------+----
-----------+---------+----------+----------+
|    Random Forest     |       BOW       |         500         |
13        |  0.898  |   0.9    |  83.733  |
| Gradient Boosting DT |       BOW       |         400         |
9         |  0.932  |  0.951   |  91.21   |
|    Random Forest     |      TF-IDF     |         500         |
17        |  0.904  |  0.914   |  85.827  |
| Gradient Boosting DT |      TF-IDF     |         500         |
9         |  0.933  |  0.952   |  91.343  |
|    Random Forest     |     Avg. W2V    |         500         |
17        |  0.857  |  0.938   |  88.727  |
| Gradient Boosting DT |     Avg-W2V     |         500         |
5         |  0.878  |  0.944   |  89.767  |
|    Random Forest     | TF-IDF Avg. W2V |         500         |
17        |   0.83  |  0.939   |   88.8   |
| Gradient Boosting DT | TF-IDF Avg. W2V |         500         |
5         |  0.855  |  0.941   |  89.16   |
+----------------------+-----------------+----------------------+----
-----------+---------+----------+----------+
```

Conclusion

1) The Best ROC_AUC score was of TF-IDF featurization and GBDT technique with maximum depth of 9 and number of estimator of 500.

2) The Best Accuracy score was of TF-IDF featurization and GBDT technique with maximum depth of 9 and number of estimator of 500.

3) GBDT technique takes alot of time to train and execute when compared to Decision Tree but has increased the model score by around 3-5% in BOW and TF-IDF featurization.

```
In [0]:
```