Customer Complaints Classification

Pujan Malavia

In [ ]:
```
from IPython.display import display
from PIL import Image
path= "C:/Users/puj83/OneDrive/Portfolio/Complaints/cfpb.png"
display(Image.open(path))
```

Link to Dataset:

https://catalog.data.gov/dataset/consumer-complaint-database (https://catalog.data.gov/dataset/consumer-complaint-database)

Abstract:

Each week the American Consumer Financial Protection Bureau (CFPB) sends thousands of consumers' complaints about financial products and services to companies for response. Those complaints are published here CFPB after the company responds or after 15 days. By adding their voice, consumers help improve the financial marketplace get their voices heard.

https://www.kaggle.com/cfpb/us-consumer-finance-complaints (https://www.kaggle.com/cfpb/us-consumer-finance-complaints)

https://www.consumerfinance.gov/ (https://www.consumerfinance.gov/)

Industry:

Government Administration/Financial Services

Company Information:

The Consumer Financial Protection Bureau (CFPB) is an agency of the United States government responsible for consumer protection in the financial sector. CFPB's jurisdiction includes banks, credit unions, securities firms, payday lenders, mortgage-servicing operations, foreclosure relief services, debt collectors and other financial companies operating in the United States.

https://en.wikipedia.org/wiki/Consumer_Financial_Protection_Bureau (https://en.wikipedia.org/wiki/Consumer_Financial_Protection_Bureau) https://www.consumerfinance.gov/ (https://www.consumerfinance.gov/)

Tool:

Python (Jupyter Notebook)

Use Case:

Building a model to predict consumer disputation rate

Initial Dataset(s):

Consumer Complaints Class

Data:

The Consumer Complaint Database is a collection of complaints about consumer financial products and services that we sent to companies for response. Complaints are published after the company responds, confirming a commercial relationship with the consumer, or after 15 days, whichever comes first. Complaints referred to other regulators, such as complaints about depository institutions with less than $10 billion in assets, are not published in the Consumer Complaint Database. The database generally updates daily.

Data Fields:

Date received

Product

Sub-product

Issue

Sub-issue

Consumer complaint narrative

Company public response

Company

State

ZIP code

Tags

Consumer consent provided?

Submitted via

Date sent to company

Company response to consumer

Timely response?

Consumer disputed?

Complaint ID

In [ ]:
```
# !pip install matplotlib.pyplot
# !pip install scikit-plot
```

Importing Libraries

In [55]:
```
import pandas as pd
from pandas import DataFrame, Series
import numpy as np
import matplotlib.pyplot as plt
import datetime
import random
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
from sklearn.preprocessing import StandardScaler
import scikitplot as skplt
%matplotlib inline
```

Importing Dataset(s)

In [56]:
```
# Read dataframe into Jupyter
df = pd.read_csv('C:/Users/puj83/OneDrive/Portfolio/Complaints/complaints.csv'
)
df = df[df['Consumer disputed?'].notnull()]
```

In [57]:
```
df.head(15).T
```

Out[57]:

	175	256	289	407	556	632
Date received	2016-07-13	2017-04-13	2016-06-29	2015-05-20	2016-01-15	2016-10-17
Product	Debt collection	Credit card	Debt collection	Debt collection	Bank account or service	Debt collection
Sub-product	I do not know	NaN	Credit card	Payday loan	Savings account	Payday loan
Issue	Disclosure verification of debt	Other	Taking/threatening an illegal action	Communication tactics	Deposits and withdrawals	Cont'd attempts collect debt not owed
Sub-issue	Right to dispute notice not received	NaN	Threatened to sue on too old debt	Called after sent written cease of comm	NaN	Debt is not mine
Consumer complaint narrative	I monitor my credit report, more frequently no...	I was stupid enough to charge some items at MA...	XXXX/XXXX/XXXX I received a letter from ARA, I...	I had my vehicle repoed & I had to use XXXX pa...	NaN	Received notification of newly added collectio..
Company public response	NaN	Company has responded to the consumer and the ...	NaN	Company believes complaint represents an oppor...	NaN	Company believes i acted appropriately as aut..
Company	Midwest Recovery Systems	CITIBANK, N.A.	Financial Credit Service, Inc.	Chek Cash, Inc.	FIRSTBANK PUERTO RICO	Midwest Recovery Systems
State	VA	NC	TX	TX	PR	CA
ZIP code	NaN	286XX	NaN	770XX	00682	910XX
Tags	Servicemember	NaN	NaN	NaN	NaN	NaN
Consumer consent provided?	Consent provided	Consent provided	Consent provided	Consent provided	Consent not provided	Consent provided
Submitted via	Web	Web	Web	Web	Web	Web
Date sent to company	2016-07-13	2017-04-13	2016-06-30	2015-07-02	2016-01-21	2016-10-17
Company response to consumer	Closed with explanation	Closed with monetary relief	Closed with explanation	Closed with explanation	Closed with explanation	Closed with explanation
Timely response?	No	Yes	No	Yes	Yes	Yes

	175	256	289	407	556	632
Consumer disputed?	Yes	No	No	No	No	No
Complaint ID	2010655	2432795	1991793	1384427	1744283	2163493

In [58]:
```
df.shape
```

Out[58]:
```
(768477, 18)
```

In [59]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 768477 entries, 175 to 1650533
Data columns (total 18 columns):
Date received                   768477 non-null object
Product                         768477 non-null object
Sub-product                     533312 non-null object
Issue                           768477 non-null object
Sub-issue                       313065 non-null object
Consumer complaint narrative    164066 non-null object
Company public response         195709 non-null object
Company                         768477 non-null object
State                           762826 non-null object
ZIP code                        722849 non-null object
Tags                            108503 non-null object
Consumer consent provided?      297904 non-null object
Submitted via                   768477 non-null object
Date sent to company            768477 non-null object
Company response to consumer    768477 non-null object
Timely response?                768477 non-null object
Consumer disputed?              768477 non-null object
Complaint ID                    768477 non-null int64
dtypes: int64(1), object(17)
memory usage: 111.4+ MB
```

In [60]:
```
df['Company'].value_counts(dropna=False).shape
```

Out[60]:
```
(4289,)
```

In [61]:
```
df['Consumer disputed?'].value_counts(dropna=False)
```

Out[61]:
```
No     620099
Yes    148378
Name: Consumer disputed?, dtype: int64
```

In [62]:
```
df['Product'].value_counts(dropna=False)
```

Out[62]:
```
Mortgage                      226897
Debt collection               145815
Credit reporting              140432
Credit card                    89190
Bank account or service        86206
Student loan                   32537
Consumer Loan                  31604
Payday loan                     5543
Money transfers                 5354
Prepaid card                    3819
Other financial service         1059
Virtual currency                  18
Checking or savings account        3
Name: Product, dtype: int64
```

In [63]:
```
df['Sub-issue'].value_counts().shape
```

Out[63]:
```
(61,)
```

In [64]:
```
df['Issue'].value_counts()
```

Out[64]:

Loan modification,collection,foreclosure	112309
Incorrect information on credit report	102686
Loan servicing, payments, escrow account	77333
Cont'd attempts collect debt not owed	60682
Account opening, closing, or management	37961
Disclosure verification of debt	30796
Communication tactics	23837
Deposits and withdrawals	22851
Dealing with my lender or servicer	17630
Application, originator, mortgage broker	17229
Credit reporting company's investigation	16883
Managing the loan or lease	15283
Billing disputes	15136
Other	14779
Problems caused by my funds being low	11845
False statements or representation	11573
Unable to get credit report/credit score	10859
Improper contact or sharing of info	10068
Problems when you are unable to pay	9385
Settlement process and costs	8940
Taking/threatening an illegal action	8859
Can't repay my loan	8726
Identity theft / Fraud / Embezzlement	8481
Making/receiving payments, sending money	7404
Closing/Cancelling account	6389
Using a debit or ATM card	6145
Credit decision / Underwriting	5652
Improper use of my credit report	5580
APR or interest rate	5506
Credit monitoring or identity protection	4424
	...
Applied for loan/did not receive money	345
Sale of account	344
Shopping for a line of credit	302
Charged bank acct wrong day or amt	297
Customer service/Customer relations	283
Wrong amount charged or received	269
Cash advance	245
Fees	232
Balance transfer fee	221
Overlimit fee	215
Incorrect/missing disclosures or info	202
Adding money	202
Cash advance fee	196
Convenience checks	149
Unexpected/Other fees	103
Excessive fees	101
Lender repossessed or sold the vehicle	79
Advertising, marketing or disclosures	77
Overdraft, savings or rewards features	53
Disclosures	49
Lost or stolen money order	46
Lost or stolen check	43
Incorrect exchange rate	22
Lender damaged or destroyed vehicle	8
Lender sold the property	7
Struggling to pay mortgage	6

```
Trouble during payment process                    3
Lender damaged or destroyed property              3
Managing an account                               2
Opening an account                                1
Name: Issue, Length: 99, dtype: int64
```

In [65]:
```
df['Company public response'].value_counts(dropna=False)
```

Out[65]:
```
NaN
572768
Company has responded to the consumer and the CFPB and chooses not to provide
a public response                                 95588
Company chooses not to provide a public response
52473
Company believes it acted appropriately as authorized by contract or law
34129
Company believes the complaint is the result of a misunderstanding
3142
Company disputes the facts presented in the complaint
2860
Company believes complaint caused principally by actions of third party outsi
de the control or direction of the company      2550
Company believes complaint is the result of an isolated error
2315
Company can't verify or dispute the facts in the complaint
1544
Company believes complaint represents an opportunity for improvement to bette
r serve consumers                                 1066
Company believes complaint relates to a discontinued policy or procedure
42
Name: Company public response, dtype: int64
```

In [66]:
```
df['Company response to consumer'].value_counts(dropna=False)
```

Out[66]:
```
Closed with explanation           577893
Closed with non-monetary relief    95428
Closed with monetary relief        51381
Closed without relief              17868
Closed                             17611
Closed with relief                  5304
Untimely response                   2992
Name: Company response to consumer, dtype: int64
```

In [67]:
```
df['Tags'].value_counts(dropna=False)
```

Out[67]:
```
NaN                             659974
Older American                   61484
Servicemember                    37994
Older American, Servicemember     9025
Name: Tags, dtype: int64
```

In [68]:
```
# df['Date received'].dtype()
df['Date received']= pd.to_datetime(df['Date received'])
df['Date received'].max()
```

Out[68]:
```
Timestamp('2017-04-22 00:00:00')
```

In [69]:
```
df['Date received'].min()
```

Out[69]:
```
Timestamp('2011-12-01 00:00:00')
```

In [70]:
```
df['Submitted via'].value_counts(dropna=False)
```

Out[70]:
```
Web             523078
Referral        133252
Phone            52201
Postal mail      48674
Fax              10924
Email              348
Name: Submitted via, dtype: int64
```

In [71]:
```
df['Timely response?'].value_counts(dropna=False)
```

Out[71]:
```
Yes    746962
No      21515
Name: Timely response?, dtype: int64
```

In [72]:
```
df['Consumer disputed?'].value_counts(dropna=False)
```

Out[72]:
```
No     620099
Yes    148378
Name: Consumer disputed?, dtype: int64
```

In [73]:
```
df["Company"]= df["Company"].astype(str)
df["Consumer disputed?"]= df["Consumer disputed?"].astype(str)
```

In [74]:
```
company_complaint_counts = df['Company'].value_counts()
df['company_complaint_counts'] = df['Company'].apply(lambda x: company_complai
nt_counts[x])
```

In [75]:
```
company = pd.crosstab(df['Company'], df['Consumer disputed?'])
```

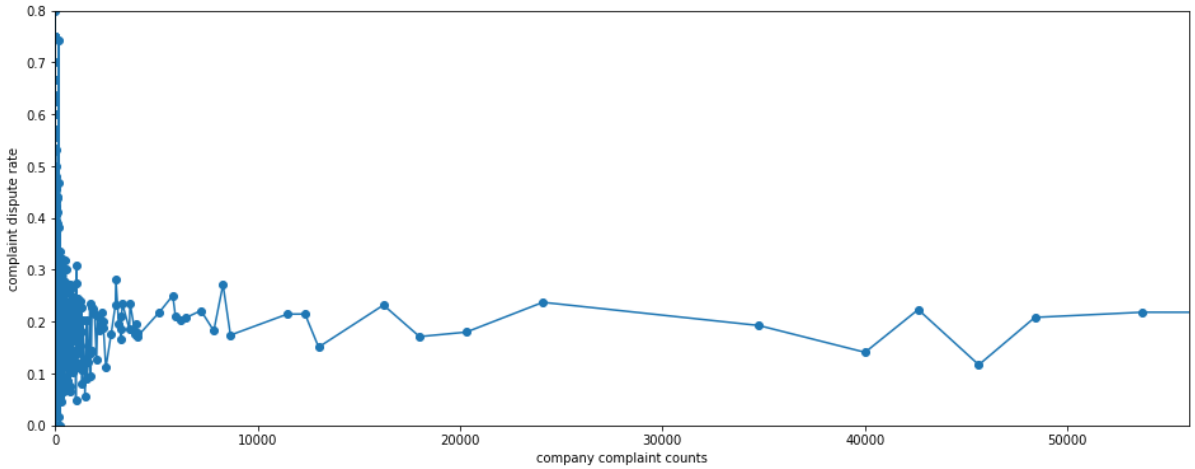
In [76]:
```
company['dispute_rate'] = company.Yes / (company.Yes + company.No)
```

In [77]:
```
company['company'] = company.index
```

In [78]:
```
company['complaint_counts'] = company['company'].apply(lambda x: company_compl
aint_counts[x])
```

In [79]:
```
company = company.sort_values('complaint_counts', ascending=False)
```

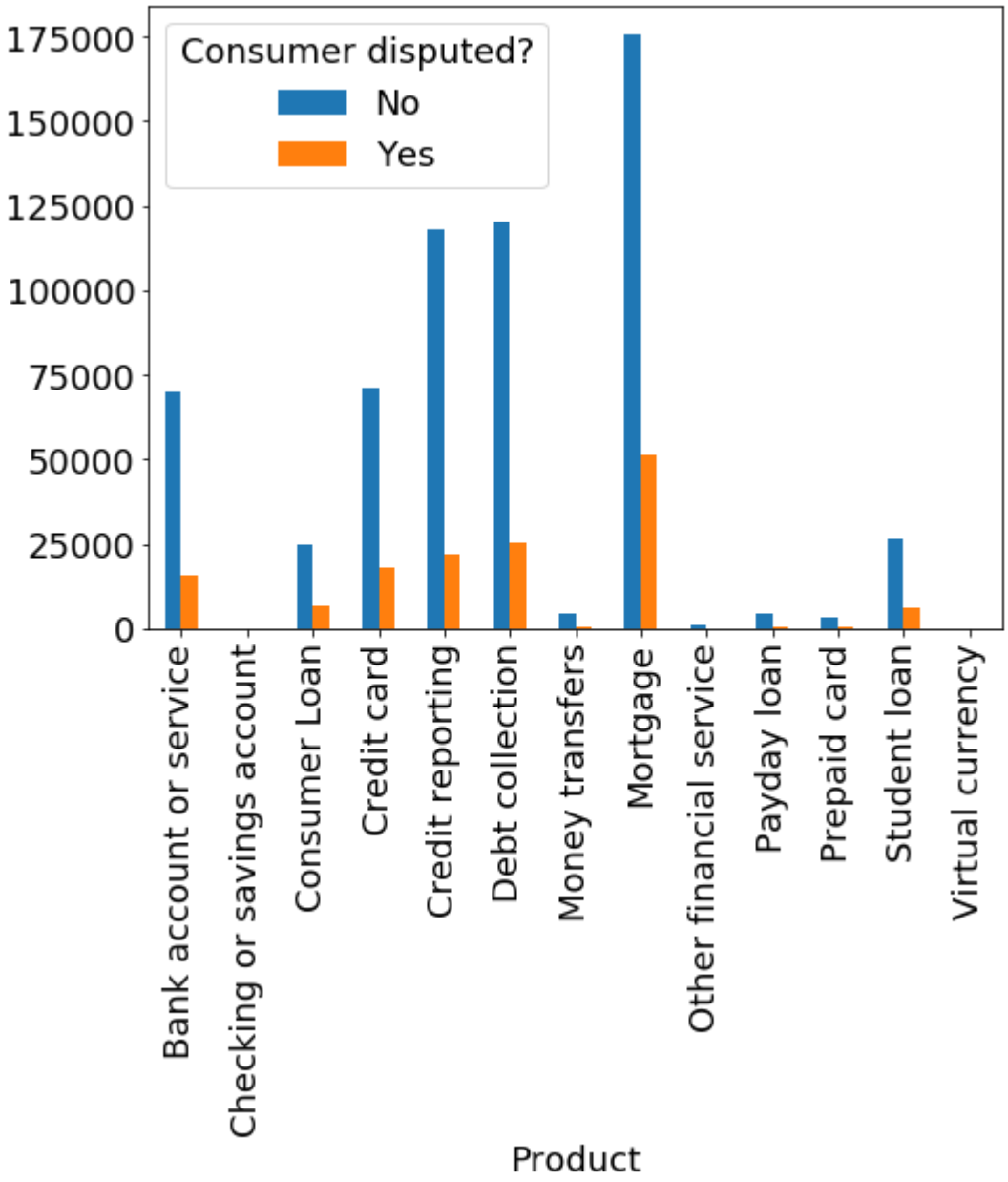
In [80]:
```
fig = plt.figure(figsize=(16,6))
plt.plot(company['complaint_counts'],company['dispute_rate'],marker='o')
plt.xlim([0.0, 56000])
plt.ylim([0.0, 0.8])
plt.xlabel('company complaint counts')
plt.ylabel('complaint dispute rate')
plt.title('')
font = {'family' : 'normal',
        'weight' : 'bold',
        'size'   : 22}
plt.rcParams.update({'font.size': 18})

plt.show()
fig.savefig('disputerate_complaintcount.png')
```



In [81]:
```
temp= pd.crosstab(df['Product'], df['Consumer disputed?'])
```

In [82]:
```
temp.plot(kind='bar',figsize=(8,6))## The disputed percentages are about same
 between
###Consent and Consent Not "complaint narrative text".
```

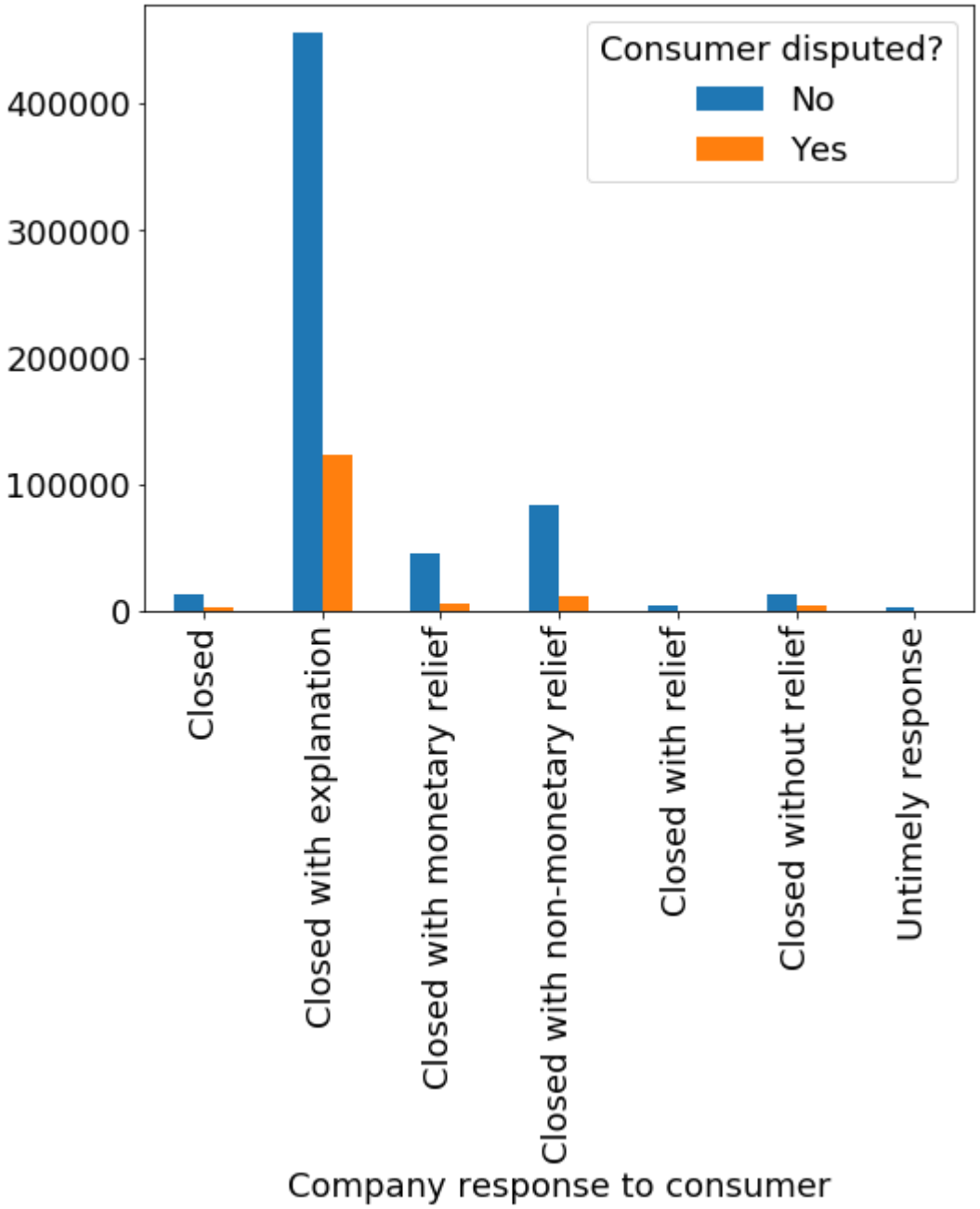
Out[82]: <matplotlib.axes._subplots.AxesSubplot at 0x1c3f0e7a508>



In [83]:
```
temp1= pd.crosstab(df['Company response to consumer'], df['Consumer disputed?'
])
```

In [84]:
```
temp1.plot(kind='bar',figsize=(8,6)) ###Most cases are fall in closed with explaination
```

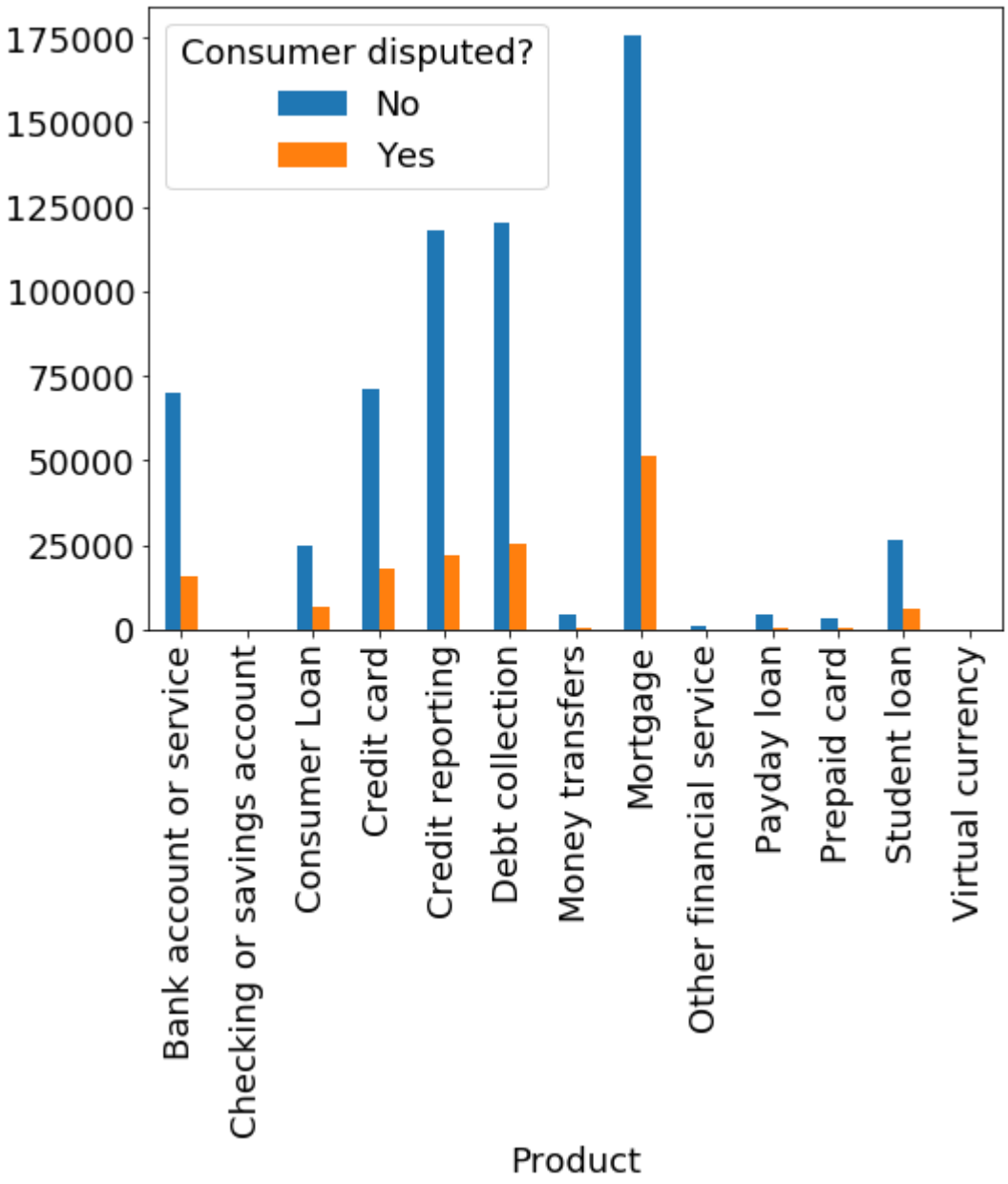
Out[84]: <matplotlib.axes._subplots.AxesSubplot at 0x1c3dee87308>



In [85]:
```
temp3= pd.crosstab(df['Product'], df['Consumer disputed?'])
```

In [86]:
```
temp3.plot(kind='bar',figsize=(8,6))
```

Out[86]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c3de53b488>
```



In [87]:
```
df['State'].value_counts().shape
```

Out[87]:
```
(62,)
```

In [88]:
```
df['Date received']= pd.to_datetime(df['Date received'])
df['Date sent to company']= pd.to_datetime(df['Date sent to company'])
```

In [89]:
```
df[df['Date received']!=df['Date sent to company']].shape
```

Out[89]:
```
(399563, 19)
```

In [90]:
```
df[pd.isnull(df['Issue'])]
```

Out[90]:

	Date received	Product	Sub-product	Issue	Sub-issue	Consumer complaint narrative	Company public response	Company	State	ZIP code	Tags

In [91]:
```
df['Sub-product'].fillna('Not Provided',inplace=True)
df['Sub-issue'].fillna('Not Provided',inplace=True)
df['Consumer complaint narrative'].fillna('None or Not Provided',inplace=True)
###Combine "company public missing value" with "Company chose not to provide"
df['Company public response'].fillna('Company chooses not to provide',inplace=
True)

###Combine missing value of "Issue" with "Other"
df['Issue'].fillna('Other',inplace=True)

### Replace missing vlaues of 'Tags' with "'Unknown'
df['Tags'].fillna('Unknown',inplace=True)

### Replace missing vlaues of 'Submitted via' with "'other'
df['Submitted via'].fillna('Other',inplace=True)

###Combine missing value,other,and withdrawn of "Consumer consent provided? "
###with Consumer consent not provided, since only users's complaints narrative
will be provided
### with the type of Consumer consent provided
df['Consumer consent provided?'].fillna('Consent not provided',inplace=True)
df['Consumer consent provided?']=df['Consumer consent provided?'].apply(lambda
x:
            'Consent not provided' if x=='Other' or x=='Consent withdrawn' els
e x)
```

In [92]:
```
### Fill missing 'State' info using valide zipcode.
from pyzipcode import ZipCodeDatabase
zip=ZipCodeDatabase()
for i in df[pd.isnull(df['State'])&pd.notnull(df['ZIP code'])].index:
    try:
        df['State'][i]=str(zip[df['ZIP code'][i]].state)
    except:
        continue
```

```
C:\Users\puj83\anaconda3\lib\site-packages\ipykernel_launcher.py:6: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
```

In [93]:
```
df[pd.isnull(df['State'])&pd.isnull(df['ZIP code'])].shape ###Still 4268 users has no state info
```

Out[93]:
```
(5636, 19)
```

In [94]:
```
df['State'].fillna('Not provided',inplace=True)
df['ZIP code'].fillna('Not Provided',inplace=True)
```

In [95]:
```
df['Consumer consent provided?'].value_counts(dropna=False)
```

Out[95]:
```
Consent not provided    604383
Consent provided        164094
Name: Consumer consent provided?, dtype: int64
```

In [96]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 768477 entries, 175 to 1650533
Data columns (total 19 columns):
Date received                   768477 non-null datetime64[ns]
Product                         768477 non-null object
Sub-product                     768477 non-null object
Issue                           768477 non-null object
Sub-issue                       768477 non-null object
Consumer complaint narrative    768477 non-null object
Company public response         768477 non-null object
Company                         768477 non-null object
State                           768477 non-null object
ZIP code                        768477 non-null object
Tags                            768477 non-null object
Consumer consent provided?      768477 non-null object
Submitted via                   768477 non-null object
Date sent to company            768477 non-null datetime64[ns]
Company response to consumer    768477 non-null object
Timely response?                768477 non-null object
Consumer disputed?              768477 non-null object
Complaint ID                    768477 non-null int64
company_complaint_counts        768477 non-null int64
dtypes: datetime64[ns](2), int64(2), object(15)
memory usage: 137.3+ MB
```

In [97]:
```
df.head()
```

Out[97]:

	Date received	Product	Sub-product	Issue	Sub-issue	Consumer complaint narrative	Company public response	
175	2016-07-13	Debt collection	I do not know	Disclosure verification of debt	Right to dispute notice not received	I monitor my credit report, more frequently no...	Company chooses not to provide	
256	2017-04-13	Credit card	Not Provided	Other	Not Provided	I was stupid enough to charge some items at MA...	Company has responded to the consumer and the ...	(
289	2016-06-29	Debt collection	Credit card	Taking/threatening an illegal action	Threatened to sue on too old debt	XXXX/XXXX/XXXX I received a letter from ARA, I...	Company chooses not to provide	Se
407	2015-05-20	Debt collection	Payday loan	Communication tactics	Called after sent written cease of comm	I had my vehicle repoed & I had to use XXXX pa...	Company believes complaint represents an oppor...	C
556	2016-01-15	Bank account or service	Savings account	Deposits and withdrawals	Not Provided	None or Not Provided	Company chooses not to provide	FI

In [98]:
```
df.groupby(df['Consumer disputed?'])['Date received']
```

Out[98]:
```
<pandas.core.groupby.groupby.SeriesGroupBy object at 0x000001C3D3490D88>
```

In [99]:
```
temp5 = pd.crosstab(df['Consumer consent provided?'], df['Consumer disputed?'
])
```

In [100]:
```
df['Consumer consent provided?'].value_counts()
```

Out[100]:
```
Consent not provided    604383
Consent provided        164094
Name: Consumer consent provided?, dtype: int64
```

In [101]:
```
replace={'Yes':True, 'No':False}
```

In [102]:
```
df['Consumer disputed?']= df['Consumer disputed?'].apply(lambda x: replace[x])
```

In [103]:
```
#Create a dataframe including all the features in the model
df_model = DataFrame()
```

In [104]:
```
#Create 'consent provided' Boolean feature
replace1={'Consent provided':True, 'Consent not provided':False}
df_model['Consumer consent provided?']= df['Consumer consent provided?'].apply
(lambda x: replace1[x])
```

In [105]:
```
#Create the number of compliants of each company as a feature
company_complaitns_counts = df['Company'].value_counts()
df_model['company_complaint_counts'] = df['Company'].apply(lambda x: company_c
omplaitns_counts[x])
```

In [106]:
```
feature_for_model=['Product', 'Sub-product','Issue','Sub-issue', 'Company publ
ic response','Tags',
                   'Company response to consumer', 'State']
for name in feature_for_model:
    repl={}
    i=0
    for value in df[name].unique():
        repl[value] = i
        i+=1

    df[name] = df[name].apply(lambda x: repl[x])
    df_model[name] = df[name].astype('category')
```

In [107]:
```
##process time refers to days between the date CFPB received complaitns and th
e date
##when complaints were sent to company on behal of comsume
df['Process time']=(df['Date sent to company']-df['Date received']).astype('ti
medelta64[D]').astype(int)
df_model=pd.concat([df_model,df['Process time']],axis=1)
```

In [108]:
```
#Create 'Timely response" boolean feature
df['Timely response?'] = df['Timely response?'].apply(lambda x: replace[x])
df_model=pd.concat([df_model,df['Timely response?']],axis=1)
```

In [109]:
```
##Create features about complaint submitted time
df_model['Date_received_year'] = df['Date received'].apply(lambda x: x.year)
df_model['Date_received_month'] = df['Date received'].apply(lambda x: x.month)
df_model['Date_received_day'] = df['Date received'].apply(lambda x: x.day)
```

In [110]:
```
df_model.head().T
```

Out[110]:

	175	256	289	407	556
Consumer consent provided?	True	True	True	True	False
company_complaint_counts	363	34768	423	3	209
Product	0	1	0	0	2
Sub-product	0	1	2	3	4
Issue	0	1	2	3	4
Sub-issue	0	1	2	3	1
Company public response	0	1	0	2	0
Tags	0	1	1	1	1
Company response to consumer	0	1	0	0	0
State	0	1	2	2	3
Process time	0	0	1	43	6
Timely response?	False	True	False	True	True
Date_received_year	2016	2017	2016	2015	2016
Date_received_month	7	4	6	5	1
Date_received_day	13	13	29	20	15

In [111]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
import sklearn.metrics as skm
from sklearn.pipeline import Pipeline
from scipy import interp
from nltk import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.linear_model import SGDClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [112]:
```
X = df_model.values
y = df['Consumer disputed?'].values
```

In [126]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, rand
om_state=67)
```

In [127]:
```
X_train
```

Out[127]:
```
array([[False, 53693, 2, ..., 2016, 7, 29],
       [False, 227, 4, ..., 2012, 8, 20],
       [False, 8666, 4, ..., 2015, 12, 3],
       ...,
       [False, 45622, 6, ..., 2014, 11, 4],
       [True, 42630, 2, ..., 2016, 7, 1],
       [False, 2052, 0, ..., 2017, 2, 6]], dtype=object)
```

In [128]:
```
X_test
```

Out[128]:
```
array([[False, 5946, 3, ..., 2012, 5, 2],
       [True, 65992, 4, ..., 2016, 6, 8],
       [False, 40006, 6, ..., 2016, 8, 26],
       ...,
       [False, 13020, 1, ..., 2016, 5, 31],
       [False, 16222, 4, ..., 2014, 5, 12],
       [False, 5946, 2, ..., 2012, 12, 7]], dtype=object)
```

In [129]:
```
y_train
```

Out[129]:
```
array([False, False, False, ..., False, False, False])
```

In [130]:
```
y_test
```

Out[130]:
```
array([ True, False, False, ..., False,  True, False])
```

In [131]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [134]:
```
def roc_curve(probabilities, labels):
    '''
    INPUT: numpy array, numpy array
    OUTPUT: list, list, list

    Take a numpy array of the predicted probabilities and a numpy array of the
    true labels.
    Return the True Positive Rates, False Positive Rates and Thresholds for th
e
    ROC curve.
    '''

    thresholds = np.sort(probabilities)

    tprs = []
    fprs = []

    num_positive_cases = sum(labels)
    num_negative_cases = len(labels) - num_positive_cases

    for threshold in thresholds:
        # With this threshold, give the prediction of each instance
        predicted_positive = probabilities >= threshold
        # Calculate the number of correctly predicted positive cases
        true_positives = np.sum(predicted_positive * labels)
        # Calculate the number of incorrectly predicted positive cases
        false_positives = np.sum(predicted_positive) - true_positives
        # Calculate the True Positive Rate
        tpr = true_positives / float(num_positive_cases)
        # Calculate the False Positive Rate
        fpr = false_positives / float(num_negative_cases)

        fprs.append(fpr)
        tprs.append(tpr)

    return tprs, fprs, thresholds.tolist()

def plot_roc(v_probs, y_test, title, xlabel, ylabel):
    # ROC
    fig = plt.figure(figsize = (8,6))
    tpr, fpr, thresholds = roc_curve(v_probs, y_test)

    import sklearn.metrics as skm
    auc = skm.roc_auc_score(y_test, v_probs)

    plt.hold(True)
    plt.plot(fpr, tpr)

    # 45 degree line
    xx = np.linspace(0, 1.0, 20)
    plt.plot(xx, xx, 'k--')

    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
```

```
    plt.show()
    fig.savefig(title+'.png')
```

In [143]:
```
## Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=10, n_jobs=-1, class_weight='balance
d', max_features=1)
rfc.fit(X_train, y_train)
```

Out[143]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight='balance
d',
                       criterion='gini', max_depth=None, max_features=1,
                       max_leaf_nodes=None, max_samples=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=10, n_jobs=
-1,
                       oob_score=False, random_state=None, verbose=0,
                       warm_start=False)
```

In [144]:
```
rfc.score(X_test, y_test)
```

Out[144]:
```
0.7956290339371226
```

In [145]:
```
pd.crosstab(y_test, rfc.predict(X_test))
```

Out[145]:

col_0	False	True
row_0		
False	120503	3834
True	27577	1782

In [146]:
```
skm.roc_auc_score(y_test, rfc.predict_proba(X_test)[:, 1])
```

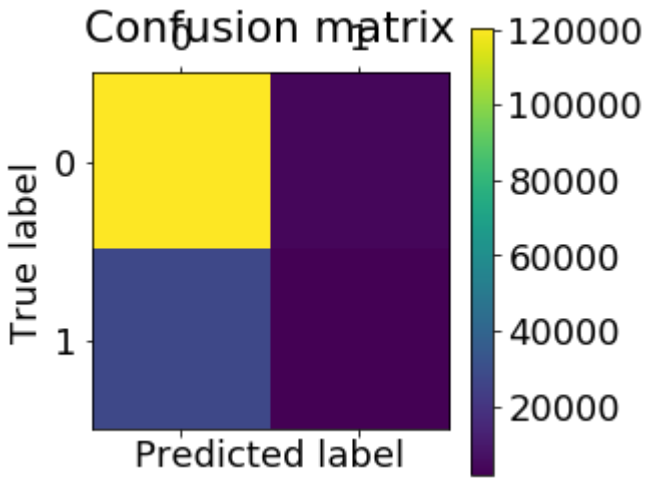
Out[146]:
```
0.5855751872953389
```

In [147]:
```
skm.recall_score(y_test, rfc.predict(X_test))
```

Out[147]:
```
0.06069689022105657
```

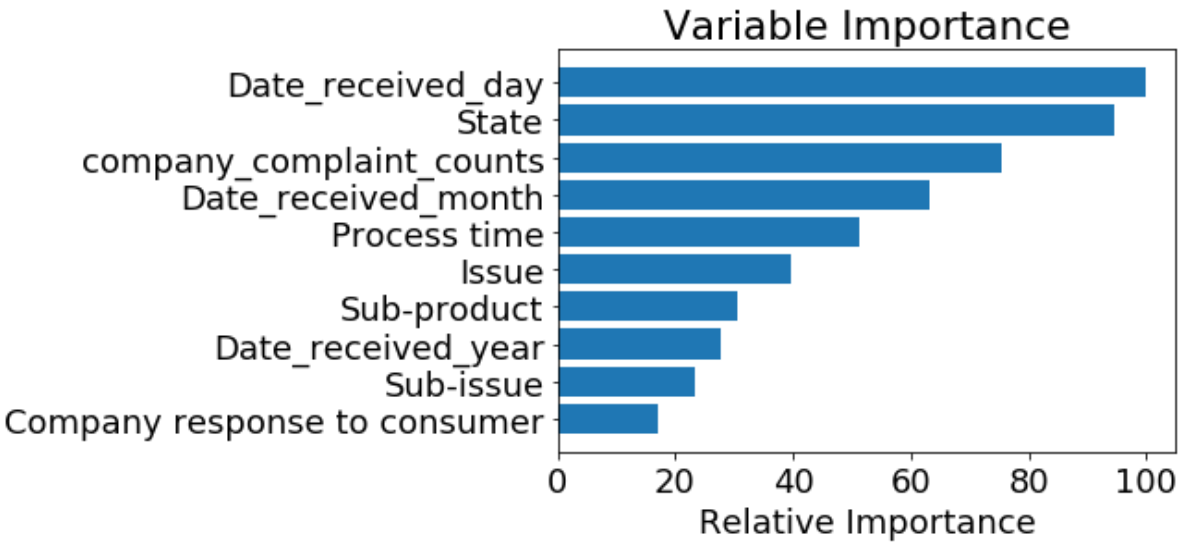
In [148]:
```
plot_confusion_matrix(rfc, X_test, y_test)
```

```
[[120503   3834]
 [ 27577   1782]]
```



In [149]:
```
def plot_importance(clf, X, max_features=10):
    '''Plot feature importance'''
    feature_importance = clf.feature_importances_
    # make importances relative to max importance
    feature_importance = 100.0 * (feature_importance / feature_importance.max
())
    sorted_idx = np.argsort(feature_importance)
    pos = np.arange(sorted_idx.shape[0]) + .5

    # Show only top features
    pos = pos[-max_features:]
    feature_importance = (feature_importance[sorted_idx])[-max_features:]
    feature_names = (X.columns[sorted_idx])[-max_features:]

    plt.barh(pos, feature_importance, align='center')
    plt.yticks(pos, feature_names)
    plt.xlabel('Relative Importance')
    plt.title('Variable Importance')
```

In [150]:
```
plot_importance(rfc, df_model, max_features=10)
```



In [151]:
```
#Create features about 'Consumenr complaint narrative'

from string import punctuation, ascii_letters

def process_text_field(text):
    '''
    text: string
    OUTPUT: int, int, int, float (length, word count, uppercase_count_rate, pu
nctuation_rate)
    '''
    length = len(text)
    word_count = 0
    last_char = False
    for c in text:
        if c in ascii_letters:
            if last_char==False:
                word_count += 1
                last_char=True
        else:
            last_char = False

    punct_count = 0
    uppercase_count = 0
    for c in text:
        if c in punctuation:
            punct_count += 1
        if c.isupper():
            uppercase_count += 1
    punctuation_rate = punct_count / float(length+1)
    uppercase_count_rate = uppercase_count / float(length+1)

    return length, word_count, uppercase_count_rate, punctuation_rate

def process_text_column(df, fieldname):
    length_list = []
    word_count_list = []
    punctuation_rate_list = []
    uppercase_count_rate_list=[]
    for row_ix in df.index:
        length, word_count, uppercase_count_rate, punctuation_rate = process_t
ext_field(df[fieldname][row_ix])
        length_list.append(length)
        word_count_list.append(word_count)
        uppercase_count_rate_list.append(uppercase_count_rate)
        punctuation_rate_list.append(punctuation_rate)
    return length_list, word_count_list, uppercase_count_rate_list, punctuatio
n_rate_list
```

In [152]:
```
stemmer = SnowballStemmer("english")

def stem_tokens(tokens, stemmer):
    stemmed=[]
    for item in tokens:
        stemmed.append(stemmer.stem(item))
    return stemmed

def tokenize(text):
    tokens = word_tokenize(text)
    stems = stem_tokens(tokens, stemmer)
    return stems
```

In [153]:
```
df=df[df['Consumer complaint narrative']!= 'None or Not Provided']
```

In [155]:
```
X_word = df['Consumer complaint narrative'].values
y = df['Consumer disputed?'].values
```

In [156]:
```
X_word.shape
```

Out[156]: (164066,)

In [157]:
```
X_train_word, X_test_word, y_train, y_test = train_test_split(X_word, y, test_
size=0.20, random_state=67)
vectorizer = TfidfVectorizer(stop_words='english',lowercase=False, min_df=0.00
1, max_df = 0.2,
                             )
words_matrix_train = vectorizer.fit_transform(X_train_word)
words_matrix_test = vectorizer.transform(X_test_word)
words_matrix = vectorizer.transform(X_word)
```

In [158]:
```
words_matrix.shape
```

Out[158]: (164066, 5108)

In [159]:
```
#MultinomialNB classifier directly using tfidf text

model = MultinomialNB()
model.fit(words_matrix_train, y_train)

##Model evaluation via 'Consumer complaint narrative' only

model.score(words_matrix_test,y_test)
skm.roc_auc_score(y_test, model.predict_proba(words_matrix_test)[:, 1])
```

Out[159]: 0.6121915943963939

In [161]:
```
# random forest directly using tfidf
rfc = RandomForestClassifier(n_estimators=1, n_jobs=-1, class_weight='balanced' )
rfc.fit(words_matrix_train, y_train)
skm.roc_auc_score(y_test, rfc.predict_proba(words_matrix_test)[:, 1])
```

Out[161]: 0.5375460203991602

In [ ]:
```
#plot_importance(rfc, words_matrix, max_features=10)
```

In [162]:
```
y_test.shape
```

Out[162]: (32814,)

In [163]:
```
v_prob = rfc.predict_proba(words_matrix_test)[:, 1]
```

In [164]:
```
v_prob.shape
```

Out[164]: (32814,)

In [166]:
```
features = np.array(vectorizer.get_feature_names())
```

In [167]:
```
type(features)
```

Out[167]: numpy.ndarray

In [168]:
```
table = DataFrame(words_matrix.toarray(), columns = features)
```

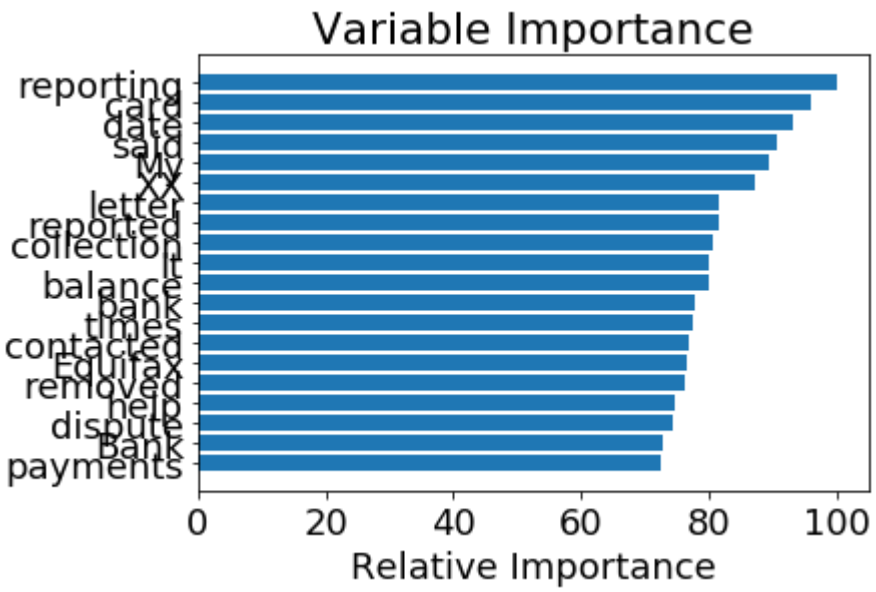
In [169]:
```
table.head()
```

Out[169]:

	000	10	100	1000	10000	100000	11	110	1100	11000	...	yearly	years	yelled	yell
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.00000	0.000
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.00000	0.000
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.14239	0.138
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.00000	0.000
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.00000	0.000

5 rows × 5108 columns

In [170]:
```
plot_importance(rfc, table, max_features=20)
font = {'family' : 'normal',
        'weight' : 'bold',
        'size'   : 22}
plt.rcParams.update({'font.size': 12})
```



In [171]:
```
from sklearn.linear_model import SGDClassifier
```

In [172]:
```
sgd = SGDClassifier(loss= 'log')
```

In [173]:
```
sgd.fit(words_matrix_train, y_train)
```

Out[173]:
```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=Tru
e,
              l1_ratio=0.15, learning_rate='optimal', loss='log', max_iter=10
00,
              n_iter_no_change=5, n_jobs=None, penalty='l2', power_t=0.5,
              random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

In [174]:
```
skm.roc_auc_score(y_test, sgd.predict_proba(words_matrix_test)[:, 1])
```

Out[174]:
```
0.6334025572042346
```

In [176]:
```
#prediction according to threshold value.
def model_predict(probs, threshold):
    pred = []
    for prob in probs:
        if prob>=threshold:
            pred.append(1)
        else: pred.append(0)
    return np.array(pred)
```

In [177]:
```
probs = sgd.predict_proba(words_matrix_test)[:, 1]
```

In [178]:
```
probs.shape
```

Out[178]:
```
(32814,)
```

In [179]:
```
predict = np.array(model_predict(probs, 0.2))
```

In [180]:
```
len(predict)
```

Out[180]:
```
32814
```

In [181]:
```
pd.crosstab(y_test,predict)
```

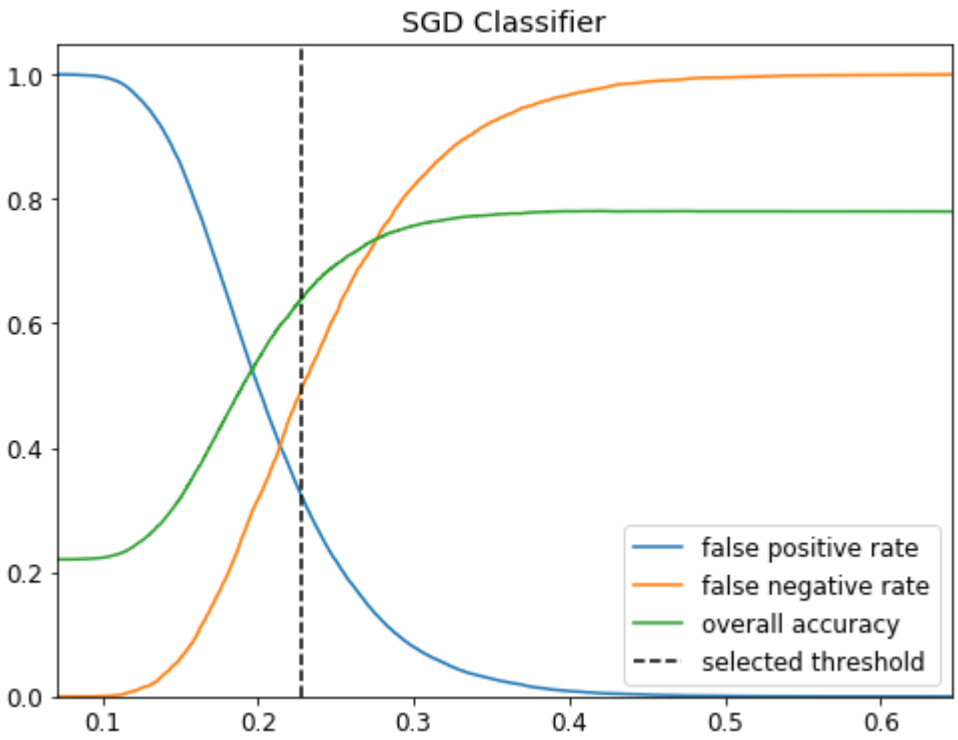
Out[181]:

col_0	0	1
row_0		
False	12832	12752
True	2301	4929

In [182]:
```
def plot_curve(probabilities, labels, title):
    '''
    INPUT: numpy array, numpy array
    OUTPUT: list, list, list

    Take a numpy array of the predicted probabilities and a numpy array of the
    true labels.
    Return the True Positive Rates, False Positive Rates and Thresholds for th
e
    ROC curve.
    '''

    thresholds = np.sort(probabilities)

    tprs = []
    fprs = []
    fnrs = []
    accus = []

    num_positive_cases = sum(labels)
    num_negative_cases = len(labels) - num_positive_cases


    for threshold in thresholds:
        # With this threshold, give the prediction of each instance
        predicted_positive = probabilities >= threshold
        predicted = np.array([1 if i >= threshold else 0 for i in probabilitie
s])
        # Calculate the number of correctly predicted positive cases
        true_positives = np.sum(predicted_positive * labels)
        # Calculate the number of incorrectly predicted positive cases
        false_positives = np.sum(predicted_positive) - true_positives
        # Calculate the True Positive Rate
        tpr = true_positives / float(num_positive_cases)
        # Calculate the False Positive Rate
        fpr = false_positives / float(num_negative_cases)
        # Calculate the False Negative Rate
        fnr = 1 - tpr
        # Calculate the overall accuracy
        accu = float(np.sum(predicted==labels))/len(labels)

        fprs.append(fpr)
        tprs.append(tpr)
        fnrs.append(fnr)
        accus.append(accu)

    fig = plt.figure(figsize=(8,6))
    plt.xlim([np.min(probabilities), np.max(probabilities)])
    plt.ylim([0.0, 1.05])
    plt.plot(thresholds.tolist(),fprs,  label = 'false positive rate')
    plt.plot(thresholds.tolist(),fnrs, label = 'false negative rate')
    plt.plot(thresholds.tolist(),accus, label = 'overall accuracy')
    plt.plot(np.repeat([0.228],100), np.linspace(0,1.05,100), 'k--', label =
'selected threshold')
    plt.legend(loc="lower right")
    plt.title(title)
```

```
    plt.show()
    fig.savefig('threshold_select.png')
```

In [183]:
```
plot_curve(probs, y_test, 'SGD Classifier')
```



In [ ]:
```

```