Text classification (multiclass)

Tutorial for beginners in text classification analysis using Python

Data: Consumer complaints received about financial products and services
Data source: Public dataset downloaded from https://catalog.data.gov/dataset/consumer-complaint-database (https://catalog.data.gov/dataset/consumer-complaint-database) on 2019, May 13.

These are real world complaints received about financial products and services. Each complaint has been labeled with a specific product; therefore, this is a supervised text classification problem. With the aim to classify future complaints based on its content, we used different machine learning algorithms can make more accurate predictions (i.e., classify the complaint in one of the product categories).

Table of Content

Goal:

Classify consumer complaints into predefined categories.

Classification algorithms: Linear Support Vector Machine (LinearSVM), Random Forest, Multinomial Naive Bayes and Logistic Regression.

Note: Text classification is an example of supervised machine learning since we train the model with labelled data (complaints about and specific finance product is used for train a classifier.

Importing packages and loading data

In [2]:

```
# Input data files are available in the "../input/" directory.
import os
print(os.listdir("C:/Users/rajul/Desktop/Coursera/Multi_class text classification"))

import pandas as pd
import numpy as np
from scipy.stats import randint
import seaborn as sns # used for plot interactive graph.
import matplotlib.pyplot as plt
import seaborn as sns
from io import StringIO
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import chi2
from IPython.display import display
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn import metrics
#import warnings
#warnings.filterwarnings("ignore", category=FutureWarning)
```

```
['.ipynb_checkpoints', 'multi-class-text-classification-tfidf.ipynb', 'rows.
csv']
```

In [3]:

```
# loading data
df = pd.read_csv('C:/Users/rajul/Desktop/Coursera/Multi_class text classification/rows.csv'
df.shape
```

```
C:\Users\rajul\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:
3165: DtypeWarning: Columns (4,5,6,11,16) have mixed types.Specify dtype opt
ion on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

Out[3]:

```
(1282355, 18)
```

We have more than 1 million instances (rows) and 18 features (columns).

Exploratory Data Analysis (EDA) and Feature Engineering

In [5]:

```
df.head(2).T # Columns are shown in rows for easy reading
```

Out[5]:

	0	1
Date received	05/10/2019	05/10/2019
Product	Checking or savings account	Checking or savings account
Sub-product	Checking account	Other banking product or service
Issue	Managing an account	Managing an account
Sub-issue	Problem using a debit or ATM card	Deposits and withdrawals
Consumer complaint narrative	NaN	NaN
Company public response	NaN	NaN
Company	NAVY FEDERAL CREDIT UNION	BOEING EMPLOYEES CREDIT UNION
State	FL	WA
ZIP code	328XX	98204
Tags	Older American	NaN
Consumer consent provided?	NaN	NaN
Submitted via	Web	Referral
Date sent to company	05/10/2019	05/10/2019
Company response to consumer	In progress	Closed with explanation
Timely response?	Yes	Yes
Consumer disputed?	NaN	NaN
Complaint ID	3238275	3238228

The dataset contains features that are not necessary to solve our multi-classification problem. For this text classification problem, we are going to build another dataframe that contains ‘Product’ and ‘Consumer complaint narrative’ (renamed as 'Consumer_complaint').

In [6]:

```
# Create a new dataframe with two columns
df1 = df[['Product', 'Consumer complaint narrative']].copy()

# Remove missing values (NaN)
df1 = df1[pd.notnull(df1['Consumer complaint narrative'])]

# Renaming second column for a simpler name
df1.columns = ['Product', 'Consumer_complaint']

df1.shape
```

Out[6]:

```
(383564, 2)
```

In [7]:

```
# Percentage of complaints with text
total = df1['Consumer_complaint'].notnull().sum()
round((total/len(df)*100),1)
```

Out[7]:

```
29.9
```

From more than 1 million complaints, there are about 380,000 cases with text (~ 30% of the original dataset is not null). This is still a good number to work with. Now let's have a look at the categories we want to classify each complaint.

In [8]:

```
pd.DataFrame(df.Product.unique()).values
```

Out[8]:

```
array([['Checking or savings account'],
       ['Debt collection'],
       ['Credit reporting, credit repair services, or other personal consume
r reports'],
       ['Mortgage'],
       ['Student loan'],
       ['Vehicle loan or lease'],
       ['Credit card or prepaid card'],
       ['Payday loan, title loan, or personal loan'],
       ['Money transfer, virtual currency, or money service'],
       ['Credit reporting'],
       ['Credit card'],
       ['Bank account or service'],
       ['Consumer Loan'],
       ['Prepaid card'],
       ['Other financial service'],
       ['Payday loan'],
       ['Money transfers'],
       ['Virtual currency']], dtype=object)
```

There are 18 different classes or categories (target). However; it is observed that some classes are contained in others. For instance, ‘Credit card’ and ‘Prepaid card’ are contained in ‘Credit card or prepaid card’ category.

Now, imagine there is a new complaint about Credit card and we want to classify it. The algorithm can either classify this complaint as 'Credit card' or 'Credit card or prepaid' and it would be correct. Nevertheless, this would affect model performance. In order to avoid this problem, the names of some categories were renamed.

In [9]:

```
# Because the computation is time consuming (in terms of CPU), the data was sampled
df2 = df1.sample(10000, random_state=1).copy()
```

In [10]:

```
# Renaming categories
df2.replace({'Product':
             {'Credit reporting, credit repair services, or other personal consumer reports
              'Credit reporting, repair, or other',
              'Credit reporting': 'Credit reporting, repair, or other',
             'Credit card': 'Credit card or prepaid card',
             'Prepaid card': 'Credit card or prepaid card',
             'Payday loan': 'Payday loan, title loan, or personal loan',
             'Money transfer': 'Money transfer, virtual currency, or money service',
             'Virtual currency': 'Money transfer, virtual currency, or money service'}},
            inplace= True)
```

In [11]:

```
pd.DataFrame(df2.Product.unique())
```

Out[11]:

	0
0	Credit reporting, repair, or other
1	Debt collection
2	Consumer Loan
3	Credit card or prepaid card
4	Mortgage
5	Vehicle loan or lease
6	Student loan
7	Payday loan, title loan, or personal loan
8	Checking or savings account
9	Bank account or service
10	Money transfer, virtual currency, or money ser...
11	Money transfers
12	Other financial service

The number of classes were reduced from 18 to 13.

Now we need to represent each class as a number, so as our predictive model can better understand the different categories.

In [12]:

```
# Create a new column 'category_id' with encoded categories
df2['category_id'] = df2['Product'].factorize()[0]
category_id_df = df2[['Product', 'category_id']].drop_duplicates()


# Dictionaries for future use
category_to_id = dict(category_id_df.values)
id_to_category = dict(category_id_df[['category_id', 'Product']].values)

# New dataframe
df2.head()
```

Out[12]:

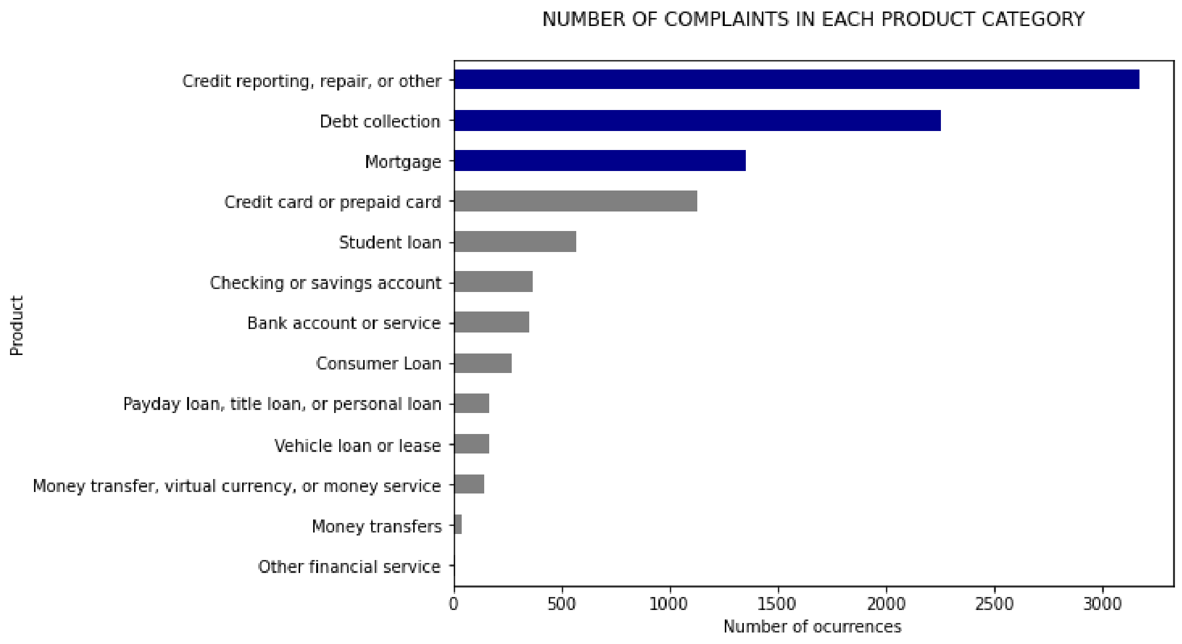
	Product	Consumer_complaint	category_id
310399	Credit reporting, repair, or other	Bayview completely ignored laws and is reporti...	0
186155	Credit reporting, repair, or other	inaccurate information no knowledge of account...	0
651135	Debt collection	On XXXX/XXXX/2016 I received a notice from Con...	1
515233	Consumer Loan	I have a loan with Kia Motor Finance. I rece...	2
641112	Credit card or prepaid card	I have a XXXX BIG card and made my payment whe...	3

The bar chart below shows the number of complaints per category. It can be observed that The bar chart below shows the number of complaints per category. It can be observed that most of customer complaints are due to:

- credit reporting, credit repair
- debt collection
- mortgage

In [13]:

```
fig = plt.figure(figsize=(8,6))
colors = ['grey','grey','grey','grey','grey','grey','grey','grey','grey',
    'grey','darkblue','darkblue','darkblue']
df2.groupby('Product').Consumer_complaint.count().sort_values().plot.barh(
    ylim=0, color=colors, title= 'NUMBER OF COMPLAINTS IN EACH PRODUCT CATEGORY\n')
plt.xlabel('Number of ocurrences', fontsize = 10);
```



Text Preprocessing

The text needs to be transformed to vectors so as the algorithms will be able make predictions. In this case it will be used the Term Frequency – Inverse Document Frequency (TFIDF) weight to evaluate **how important a word is to a document in a collection of documents**.

After removing **punctuation** and **lower casing** the words, importance of a word is determined in terms of its frequency.

“Term Frequency – Inverse Document Frequency

TF-IDF is the product of the **TF** and **IDF** scores of the term.

$$\text{TF-IDF} = \frac{\text{TF}}{\text{IDF}}$$

Term Frequency : This summarizes how often a given word appears within a document.

$$\text{TF} = \frac{\text{Number of times the term appears in the doc}}{\text{Total number of words in the doc}}$$

Inverse Document Frequency: This downscales words that appear a lot across documents. A term has a high IDF score if it appears in a few documents. Conversely, if the term is very common among documents (i.e., “the”, “a”, “is”), the term would have a low IDF score.

$$\text{IDF} = \ln\left(\frac{\text{Number of docs}}{\text{Number docs the term appears in}}\right)$$

TF-IDF are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents. The higher the TFIDF score, the rarer the term is. For instance, in a Mortgage complaint the word *mortgage* would be mentioned fairly often. However, if we look at other complaints, *mortgage* probably would not show up in many of them. We can infer that *mortgage* is most probably an important word in Mortgage complaints as compared to the other products. Therefore, *mortgage* would have a high TF-IDF score for Mortgage complaints.

TfidfVectorizer class can be initialized with the following parameters:

- **min_df**: remove the words from the vocabulary which have occurred in less than ‘min_df’ number of files.
- **max_df**: remove the words from the vocabulary which have occurred in more than *‘max_df’ * total number of files in corpus*.
- **sublinear_tf**: set to True to scale the term frequency in logarithmic scale.
- **stop_words**: remove the predefined stop words in 'english'.
- **use_idf**: weight factor must use inverse document frequency.
- **ngram_range**: (1, 2) to indicate that unigrams and bigrams will be considered.

In [14]:

```
tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5,
                        ngram_range=(1, 2),
                        stop_words='english')

# We transform each complaint into a vector
features = tfidf.fit_transform(df2.Consumer_complaint).toarray()

labels = df2.category_id

print("Each of the %d complaints is represented by %d features (TF-IDF score of unigrams an
```

```
Each of the 10000 complaints is represented by 27507 features (TF-IDF score
of unigrams and bigrams)
```

In [15]:

```
# Finding the three most correlated terms with each of the product categories
N = 3
for Product, category_id in sorted(category_to_id.items()):
  features_chi2 = chi2(features, labels == category_id)
  indices = np.argsort(features_chi2[0])
  feature_names = np.array(tfidf.get_feature_names())[indices]
  unigrams = [v for v in feature_names if len(v.split(' ')) == 1]
  bigrams = [v for v in feature_names if len(v.split(' ')) == 2]
  print("\n==> %s:" %(Product))
  print("  * Most Correlated Unigrams are: %s" %(', '.join(unigrams[-N:])))
  print("  * Most Correlated Bigrams are: %s" %(', '.join(bigrams[-N:])))
```

```

==> Bank account or service:
  * Most Correlated Unigrams are: overdraft, bank, scottrade
  * Most Correlated Bigrams are: citigold checking, debit card, checking
account

==> Checking or savings account:
  * Most Correlated Unigrams are: checking, branch, overdraft
  * Most Correlated Bigrams are: 00 bonus, overdraft fees, checking accou
nt

==> Consumer Loan:
  * Most Correlated Unigrams are: dealership, vehicle, car
  * Most Correlated Bigrams are: car loan, vehicle loan, regional accepta
nce

==> Credit card or prepaid card:
  * Most Correlated Unigrams are: express, citi, card
  * Most Correlated Bigrams are: balance transfer, american express, cred
it card

==> Credit reporting, repair, or other:
  * Most Correlated Unigrams are: report, experian, equifax
  * Most Correlated Bigrams are: credit file, equifax xxxx, credit report

==> Debt collection:
  * Most Correlated Unigrams are: collect, collection, debt
  * Most Correlated Bigrams are: debt collector, collect debt, collection
agency

==> Money transfer, virtual currency, or money service:
  * Most Correlated Unigrams are: ethereum, bitcoin, coinbase
  * Most Correlated Bigrams are: account coinbase, coinbase xxxx, coinbas
e account

==> Money transfers:
  * Most Correlated Unigrams are: paypal, moneygram, gram
  * Most Correlated Bigrams are: sending money, western union, money gram

==> Mortgage:
  * Most Correlated Unigrams are: escrow, modification, mortgage
  * Most Correlated Bigrams are: short sale, mortgage company, loan modif
ication

==> Other financial service:
  * Most Correlated Unigrams are: meetings, productive, vast
  * Most Correlated Bigrams are: insurance check, check payable, face fac
e

==> Payday loan, title loan, or personal loan:
  * Most Correlated Unigrams are: astra, ace, payday
  * Most Correlated Bigrams are: 00 loan, applied payday, payday loan

==> Student loan:
  * Most Correlated Unigrams are: student, loans, navient
  * Most Correlated Bigrams are: income based, student loan, student loan
s

==> Vehicle loan or lease:
  * Most Correlated Unigrams are: honda, car, vehicle
  * Most Correlated Bigrams are: used vehicle, total loss, honda financia
l
```

Multi-Classification models

The classification models evaluated are:

- Random Forest
- Linear Support Vector Machine
- Multinomial Naive Bayes
- Logistic Regression.

Spliting the data into train and test sets

The original data was divided into features (X) and target (y), which were then splitted into train (75%) and test (25%) sets. Thus, the algorithms would be trained on one set of data and tested out on a completely different set of data (not seen before by the algorithm).

In [16]:

```
X = df2['Consumer_complaint'] # Collection of documents
y = df2['Product'] # Target or the labels we want to predict (i.e., the 13 different compla

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.25,
                                                    random_state = 0)
```

Models

In [18]:

```
models = [
    RandomForestClassifier(n_estimators=100, max_depth=5, random_state=0),
    LinearSVC(),
    MultinomialNB(),
    LogisticRegression(random_state=0),
]

# 5 Cross-validation
CV = 5
cv_df = pd.DataFrame(index=range(CV * len(models)))

entries = []
for model in models:
  model_name = model.__class__.__name__
  accuracies = cross_val_score(model, features, labels, scoring='accuracy', cv=CV)
  for fold_idx, accuracy in enumerate(accuracies):
    entries.append((model_name, fold_idx, accuracy))

cv_df = pd.DataFrame(entries, columns=['model_name', 'fold_idx', 'accuracy'])
```

```
C:\Users\rajul\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
C:\Users\rajul\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
C:\Users\rajul\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
C:\Users\rajul\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:763: ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
C:\Users\rajul\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
```

Comparison of model performance

The best mean acuracy was obtained with LinearSVC.

In [19]:

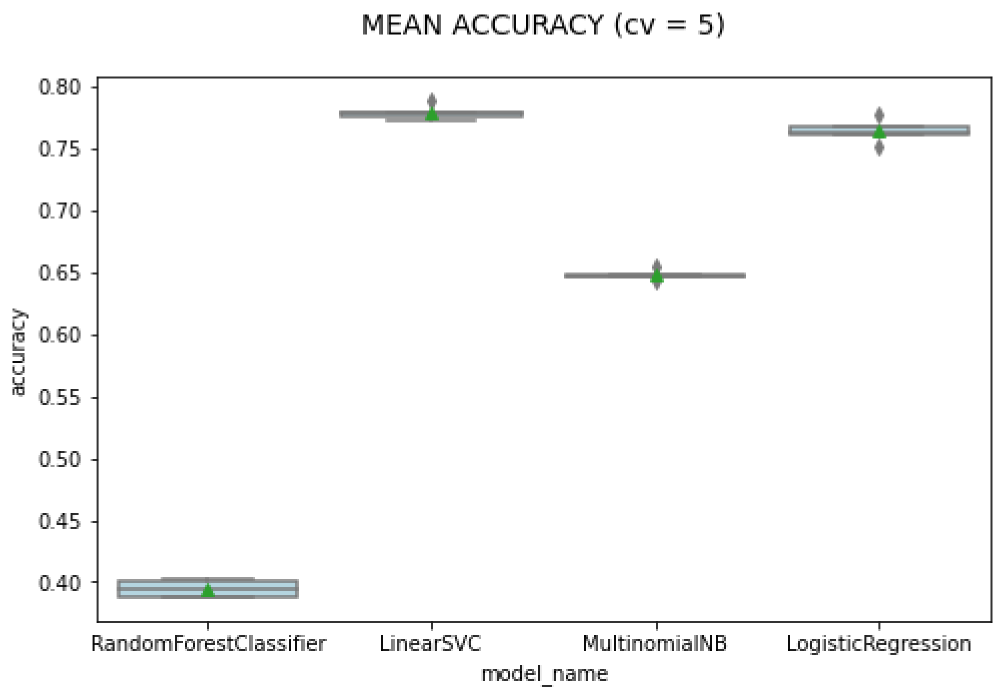
```
mean_accuracy = cv_df.groupby('model_name').accuracy.mean()
std_accuracy = cv_df.groupby('model_name').accuracy.std()

acc = pd.concat([mean_accuracy, std_accuracy], axis= 1,
          ignore_index=True)
acc.columns = ['Mean Accuracy', 'Standard deviation']
acc
```

Out[19]:

	Mean Accuracy	Standard deviation
model_name		
LinearSVC	0.7791	0.005561
LogisticRegression	0.7641	0.009839
MultinomialNB	0.6475	0.004596
RandomForestClassifier	0.3949	0.006665

In [20]:

```
plt.figure(figsize=(8,5))
sns.boxplot(x='model_name', y='accuracy',
            data=cv_df,
            color='lightblue',
            showmeans=True)
plt.title("MEAN ACCURACY (cv = 5)\n", size=14);
```



Model Evaluation

In [21]:

```
X_train, X_test, y_train, y_test,indices_train,indices_test = train_test_split(features,
                                                               labels,
                                                               df2.index, test_size=0.25,
                                                               random_state=1)

model = LinearSVC()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Precision, Recall, F1-score

In [22]:

```
# Classification report
print('\t\t\t\tCLASSIFICATIION METRICS\n')
print(metrics.classification_report(y_test, y_pred,
                                    target_names= df2['Product'].unique()))
```

```
                                 CLASSIFICATIION METRICS

                                                         precision    recall  f1-
score   support

                Credit reporting, repair, or other       0.81      0.88      
0.84       783
                                   Debt collection       0.76      0.80      
0.78       588
                                     Consumer Loan       0.52      0.22      
0.31        73
                       Credit card or prepaid card       0.73      0.82      
0.77       253
                                          Mortgage       0.87      0.92      
0.90       340
                            Vehicle loan or lease        0.43      0.14      
0.21        43
                                      Student loan       0.85      0.85      
0.85       155
         Payday loan, title loan, or personal loan       0.50      0.17      
0.26        40
                       Checking or savings account       0.61      0.60      
0.61        85
                           Bank account or service       0.61      0.43      
0.50        93
Money transfer, virtual currency, or money service       0.83      0.62      
0.71        32
                                   Money transfers       0.60      0.23      
0.33        13
                           Other financial service       0.00      0.00      
0.00         2

                                          accuracy                           
0.78      2500
                                         macro avg       0.62      0.51      
0.54      2500
                                      weighted avg       0.77      0.78      
0.77      2500


C:\Users\rajul\anaconda3\lib\site-packages\sklearn\metrics\_classificatio
n.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined
and being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rajul\anaconda3\lib\site-packages\sklearn\metrics\_classificatio
n.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined
and being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\rajul\anaconda3\lib\site-packages\sklearn\metrics\_classificatio
n.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined
```

```
and being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

It is possible to observe that the classes with more support (number of occurrences) tend to have a better f1-cscore. This is because the algorithm was trained with more data.

The classes that can be classified with more precision are **‘Mortgage’**, **‘Credit reporting, repair, or other’**, and **‘Student loan’**.

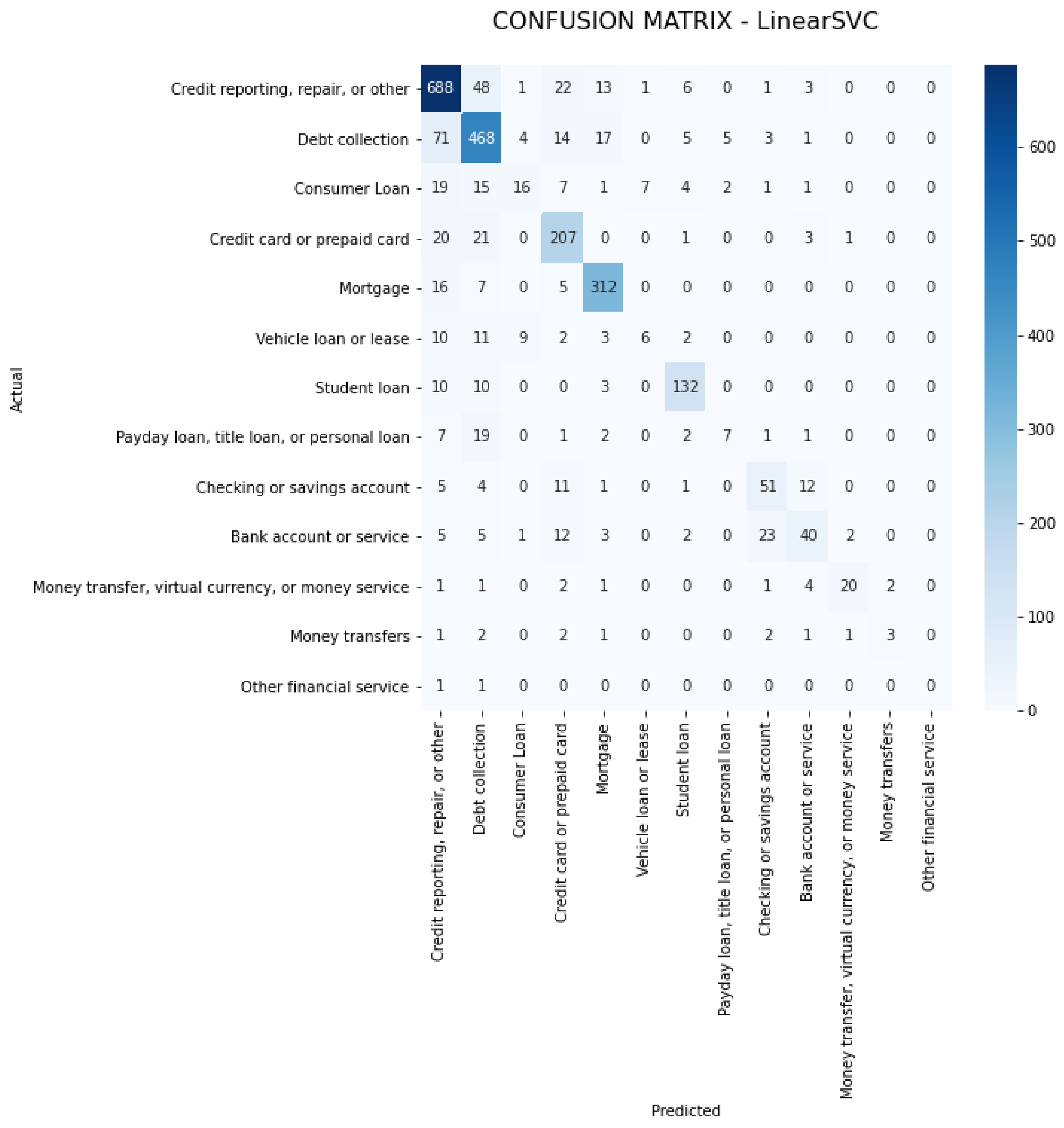
Confusion Matrix

A Confusion Matrix is a table which rows represent the actual class and columns represents the predicted class.

If we had a perfect model that always classifies correctly a new complaint, then the confusion matrix would have values in the diagonal only (where predicted label = actual label).

In [23]:

```python
conf_mat = confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots(figsize=(8,8))
sns.heatmap(conf_mat, annot=True, cmap="Blues", fmt='d',
            xticklabels=category_id_df.Product.values,
            yticklabels=category_id_df.Product.values)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title("CONFUSION MATRIX - LinearSVC\n", size=16);
```



In general, the confusion matrix looks good (clear diagonal that represents correct classifications). Nevertheless, there are cases were the complaint was classified in a wrong class.

Misclassified complaints

Let’s have a look at the cases that were wrongly classified.

In [24]:

```
for predicted in category_id_df.category_id:
  for actual in category_id_df.category_id:
    if predicted != actual and conf_mat[actual, predicted] >= 20:
      print("'{}' predicted as '{}' : {} examples.".format(id_to_category[actual],
                                                            id_to_category[predicted],
                                                            conf_mat[actual, predicted]))

      display(df2.loc[indices_test[(y_test == actual) & (y_pred == predicted)]][['Product',
                                                                                  'Consumer_complaint']])

      print('')
```

```
'Debt collection' predicted as 'Credit reporting, repair, or other' : 71
examples.
```

	Product	Consumer_complaint
483075	Debt collection	This agency has three collections listed which...
111807	Debt collection	I am writing to dispute the inaccurate data be...
376788	Debt collection	check n ' Go payday loan made me a loan and th...
570764	Debt collection	THIS AMT OWED {$8800.00} WAS TO BE PAID UNDER ...
88089	Debt collection	I am submitting this complaint for an account ...
...	...	...
155139	Debt collection	Between XX/XX/XXXX and XX/XX/XXXX, this accoun...
360018	Debt collection	On XXXX XXXX, 2017 XXXX opened up an account f...
165296	Debt collection	I have tried disputing with the bureau on seve...

Most correlated terms with each category

In [25]:

```
model.fit(features, labels)

N = 4
for Product, category_id in sorted(category_to_id.items()):
  indices = np.argsort(model.coef_[category_id])
  feature_names = np.array(tfidf.get_feature_names())[indices]
  unigrams = [v for v in reversed(feature_names) if len(v.split(' ')) == 1][:N]
  bigrams = [v for v in reversed(feature_names) if len(v.split(' ')) == 2][:N]
  print("\n==> '{}':".format(Product))
  print("  * Top unigrams: %s" %(', '.join(unigrams)))
  print("  * Top bigrams: %s" %(', '.join(bigrams)))
```

```

==> 'Bank account or service':
  * Top unigrams: bank, scottrade, deposit, suntrust
  * Top bigrams: bank xxxx, check account, debit card, xx 2016

==> 'Checking or savings account':
  * Top unigrams: bank, transactions, bonus, branch
  * Top bigrams: debit card, xxxx 2017, overdraft fees, account informati
on

==> 'Consumer Loan':
  * Top unigrams: car, furniture, title, loan
  * Top bigrams: vehicle loan, lien release, regional acceptance, xxxx co
nstantly

==> 'Credit card or prepaid card':
  * Top unigrams: card, capital, citi, amex
  * Top bigrams: american express, use card, balance transfer, annual fee

==> 'Credit reporting, repair, or other':
  * Top unigrams: equifax, experian, transunion, report
  * Top bigrams: xxxx reported, equifax xxxx, xxxx xxxx, trans union

==> 'Debt collection':
  * Top unigrams: debt, collection, collections, owe
  * Top bigrams: credit services, trying collect, violation knowledge, ac
count credit

==> 'Money transfer, virtual currency, or money service':
  * Top unigrams: coinbase, seller, money, support
  * Top bigrams: able receive, bank account, western union, money transfe
r

==> 'Money transfers':
  * Top unigrams: paypal, gram, moneygram, money
  * Top bigrams: money gram, wire transfer, reported wrong, sending money

==> 'Mortgage':
  * Top unigrams: mortgage, escrow, modification, home
  * Top bigrams: mortgage company, escrow account, home xxxx, mortgage xx
xx

==> 'Other financial service':
  * Top unigrams: vast, productive, advertised, improvement
  * Top bigrams: face face, stop payment, check payable, repair credit
```

```
==> 'Payday loan, title loan, or personal loan':
  * Top unigrams: loan, payday, apr, rate
  * Top bigrams: ad astra, loan xx, payday loan, personal loan

==> 'Student loan':
  * Top unigrams: navient, loans, school, student
  * Top bigrams: sallie mae, payments credit, student loan, income based

==> 'Vehicle loan or lease':
  * Top unigrams: vehicle, honda, lease, car
  * Top bigrams: xxxx car, payments dates, account response, honda financ
ial
```

Predictions

Now let's make a few predictions on unseen data.

In [26]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.25,
                                                    random_state = 0)

tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5,
                        ngram_range=(1, 2),
                        stop_words='english')

fitted_vectorizer = tfidf.fit(X_train)
tfidf_vectorizer_vectors = fitted_vectorizer.transform(X_train)

model = LinearSVC().fit(tfidf_vectorizer_vectors, y_train)
```

Let's see what is the classification that our model gives to this new complaint.

In [27]:

```
new_complaint = """I have been enrolled back at XXXX XXXX University in the XX/XX/XXXX. Rec
Navient for the last month. I have faxed in paperwork providing them with everything they n
phone calls for payments. Furthermore, Navient is now reporting to the credit bureaus that 
Navient needs to get their act together to avoid me taking further action. I have been enro
deferment should be valid with my planned graduation date being the XX/XX/XXXX."""
print(model.predict(fitted_vectorizer.transform([new_complaint])))
```

```
['Student loan']
```

The algorithm has classified this text as a "Student loan" complaint. Now let's look at the real label of this complaint.

In [28]:

```
df2[df2['Consumer_complaint'] == new_complaint]
```

Out[28]:

	Product	Consumer_complaint	category_id
877489	Student loan	I have been enrolled back at XXXX XXXX Univers...	6

Our model was correct, the complaint was about **Student Loan**. Note that this customer has used terms with high TFIDF score, such us **Navient** (http://).

Let's check another example.

In [29]:

```
new_complaint_2 = """Equifax exposed my personal information without my consent, as part of
In addition, they dragged their feet in the announcement of the report, and even allowed th
off stock before the announcement."""
print(model.predict(fitted_vectorizer.transform([new_complaint_2])))
```

```
['Credit reporting, repair, or other']
```

In [30]:

```
df2[df2['Consumer_complaint'] == new_complaint_2]
```

Out[30]:

	Product	Consumer_complaint	category_id
420821	Credit reporting, repair, or other	Equifax exposed my personal information withou...	0

Again, the algorithm correctly classified the caomplaint as **"Credit reporting, repair, or other"**. Note that this customer has used terms with high TFIDF score, such us **equifax, report**.

Although our model is not going to be all the time correct when classifying new complaints, it does a good job.

In [ ]:

```

```

In [ ]:

```

```