Convex Optimization

Final Project: Sentiment Analysis using SVC

Autumn 2021

Team Info

Carlos Alejandro Ramos Pérez, César Contreras González, Adrian Ramos Pérez

alejandro.ramos@iteso.mx, cesar.contreras@iteso.mx, adrian.ramos@iteso.mx

Héctor Daniel Estrada Rodríguez, Yared Ismael Flores Jiménez

daniel.estrada@iteso.mx, yared.flores@iteso.mx

Project Links

- **To get the data please refer to Amazon Reviews: Unlocked Mobile Phones**
  - Access to Kaggle dataset
- **Link to repository:**
  - Access to Jupyter notebook in Github
- **Link to interactive notebook:**
  - Access to binder interactive notebook
- **Link to PowerPoint presentation (pptx):**
  - Project Presentation

Table of Contents

Go back

1 Abstract

Amazon a world-wide known company, is the largest online retailer in the world. In that sense, the size of daily buying transactions, constitutes an incredible source of information for data geeks like us. Amazon is a company that not only keeps track of the final transactions, it also collects information before and after the final purchase. For this project, an **Amazon dataset with customer's reviews and ratings** about the smartphone has been used. The present's project idea consists to use techniques of Natural Languange Processing to run a sentiment analysis under the constraint of a supervised model that can **predict whether the customer review was a positive or a negative one**.

Go back

2 About Sentiment Analysis

Sentiment analysis, which is also called opinion mining, has been one of the most active research areas in natural language processing since early 2000. The aim of sentiment analysis is to define automatic tools able to extract subjective information from texts in natural language, such as opinions and sentiments, so as to create structured and actionable knowledge to be used by either a decision support system or a decision maker. For applications that range from marketing to customer service to clinical medicine. From the definition of sentiment analysis , "the aim of sentiment analysis is therefore to define automatic tools able to extract subjective information in order to create structured and actionable knowledge."

Go back

3 Motivation

Before the arrival of Machine Learning, its techniques and tools that implied effort to classify opinions might have been demanding in extreme, if not a nightmare. Today, tons and tons of data are daily generated containing opinions and sentiments from people regarding a broad type of topics. We all as consumers try to find the best product for our needs and backup our buying decisions in a reliable process rather than just a feeling. And since, nowadays social media and specialized blogs and other kind of source are used to externalize and share opinions over service products, all of these represent information of great potential and value. Based on all these ideas and considering the received instruction on Support Vector Machines this semester, we considered this project could represent a good exercise to set up a method of decision making to fugure out the reception and acceptance of products like smartphones.

Go back

4 Objectives

1. Will demonstrate how the use of natural language processing techniques can produce the right output for a SVM analysis for classification

2. Will produce the results obtained via SVM and contrast the accuracy of the model via a supervised approach

3. Will tune hyperparameters model to obtain the best optimals

Go back

5 Theoretical Framework

Go back

5.1 Bag of words

To accomplish the idea behind this project we need to understand that in Machine Learning algorithms cannot work with raw data text directly. Rather the text must be converted into vector numbers. In natural Language processing, a common technique for extracting features from text is to place all of the words that occur in the text in a bucket. This approach is called *bag of words* model or *BoW* for short. It is referred to as a "bag" of words because any information about the structure of the sentence is lost. The vectorization can be done in a couple of ways:

1. Using Count Vectorizer
2. Using TF-IDF Verctorizer

Count Vectorizer
The count of words is what matters. In case a word does not appear in the sample then a 0 value is assigned.
TD-IDF Vectorizer
It calculates two things:

1. Term Frequency: Number of if times the word appears in the sample
2. IDF: **Number of times the world appears in the sample / number of times the world appears** in the full dataset

Go back

5.2 Support Vector Machines

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

Source: sickit-learn web documentation of SVM

Go back

5.3 SVM Mathematical model (Classification)

Primal Problem:

$$\min_{w,b,\xi_k} \quad \frac{1}{2}w^T w + C\sum \xi_K$$

$$\text{s. t. } \; y_k\,[w^T \varphi\,(x_k) + b] \geq 1 - \xi_k\,\{x_k, y_k\}_{k=1}^{N}$$

$$y_k \,\epsilon\, \{-1, 1\}\ \varphi : R^n \rightarrow R^m,\ \text{so}\ w\,\epsilon\, R^m$$

The Lagrangian:

$$L = \frac{1}{2}w^T w + C\sum_{K=1}^{N} \xi_k - \sum_{K=1}^{N} \alpha_k(y_k\,[w^T\,\varphi(x_k) + b]) - 1 + \xi_k - \sum_{K=1}^{N} \eta_k\xi_k$$

Dual problem:

$$\max_{\alpha}\ D = -\frac{1}{2}\sum_{k,l=1}^{N} \alpha_l\alpha_k y_l y_k \varphi^T(x_l)\varphi(x_k) + \sum_{k=1}^{N} \alpha_k$$

$$\text{s. t.} \sum_{k=1}^{N} \alpha_k y_k$$

Go back

5.3.1 Hyperparameter Tunning of an SVM

Hyperparameter optimization refers to performing a search in order to discover the set of specific model configuration arguments that result in the best performance of the model on a specific dataset.

There are many ways to perform hyperparameter optimization, although modern methods, such as Bayesian Optimization, are fast and effective.

Bayesian optimization is a powerful strategy for finding the extrema of objective functions that are expensive to evaluate. It is particularly useful when these **evaluations are costly**, when one **does not have access to derivatives**, or when the problem at hand is **non-convex**.

Go back

5.4 Model Performance

The idea of building machine learning models works on a constructive feedback principle. You build a model, get feedback from metrics, make improvements and continue until you achieve a desirable accuracy. Evaluation metrics explain the performance of a model. An important aspect of evaluation metrics is their capability to discriminate among model results.

Go back

5.4.1 Some metrics to measure model performance

5.4.1.1 Confusion Matrix

A confusion matrix is an N X N matrix, where N is the number of classes being predicted. For the problem in hand, we have N=2, and hence we get a 2 X 2 matrix. Here are a few definitions, you need to remember for a confusion matrix :

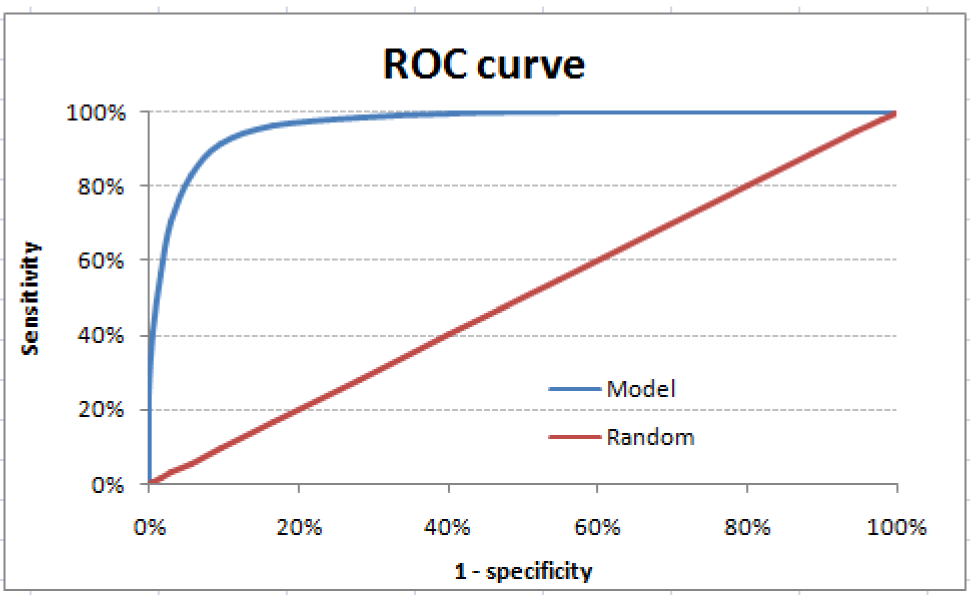
- **Accuracy** : the proportion of the total number of predictions that were correct.
- **Positive** Predictive Value or Precision : the proportion of positive cases that were correctly identified.
- **Negative** Predictive Value : the proportion of negative cases that were correctly identified.
- **Sensitivity** or Recall : the proportion of actual positive cases which are correctly identified.
- **Specificity** : the proportion of actual negative cases which are correctly identified.

Confusion Matrix		Target			
		Positive	Negative		
Model	Positive	a	b	*Positive Predictive Value*	a/(a+b)
	Negative	c	d	*Negative Predictive Value*	d/(c+d)
		Sensitivity	*Specificity*	**Accuracy** = (a+d)/(a+b+c+d)	
		a/(a+c)	d/(b+d)		

Go back

5.4.1.2 Area Under the ROC curve (AUC – ROC)

This is again one of the popular metrics used in the industry. The biggest advantage of using ROC (Receiver operating characteristic) curve is that it is independent of the change in proportion of responders. This statement will get clearer in the following sections.



The ROC curve is the plot between sensitivity and (1- specificity). (1- specificity) is also known as false positive rate and sensitivity is also known as True Positive rate

AUC itself is the ratio under the curve and the total area

Following are a few thumb rules:

- .90-1 = excellent (A)
- .80-.90 = good (B)
- .70-.80 = fair (C)
- .60-.70 = poor (D)
- .50-.60 = fail (F)

Go back

5.4.1.3 Cross Validation

Cross Validation is one of the most important concepts in any type of data modelling. It simply says, try to leave a sample on which you do not train the model and test the model on this sample before finalizing the model.

Go back

6 Descriptive Analysis

A dataset from Kaggle Web Site was employed for the purpose of this project. This dataset defines the next set of features/variables:

- Product Name
- Brand Name
- Price
- Rating
- Reviews
- Votes

Something important to point out is that the idea to keep special attention on these two variables: *Rating* and *Reviews*.

The Rating variable

- The Rating variable is a categorical variable that stores a numerical values from a scale of 1 to 5 (the 1 to 5 star rating of Amazon)

The Review variable

- The Review is the dataset feature that holds the raw text for the comments posted by every customer. This is the variable subject to vectorization

Go back

7 Predictive Analysis

Go back

I. Data Preprocessing

In [1]:
```
import sklearn.svm
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
from imblearn.over_sampling import SMOTE
```

In [3]:
```
data  = pd.read_csv('Amazon_Unlocked_Mobile.csv')
```

In [4]:
```
data
```

Out[4]:

	Product Name	Brand Name	Price	Rating	Reviews	Review Votes
0	"CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7...	Samsung	199.99	5	I feel so LUCKY to have found this used (phone...	1.0
1	"CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7...	Samsung	199.99	4	nice phone, nice up grade from my pantach revu...	0.0
2	"CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7...	Samsung	199.99	5	Very pleased	0.0
3	"CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7...	Samsung	199.99	4	It works good but it goes slow sometimes but i...	0.0
4	"CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7...	Samsung	199.99	4	Great phone to replace my lost phone. The only...	0.0
...	...	...	...	...	...	...
413835	Samsung Convoy U640 Phone for Verizon Wireless...	Samsung	79.95	5	another great deal great price	0.0
413836	Samsung Convoy U640 Phone for Verizon Wireless...	Samsung	79.95	3	Ok	0.0
413837	Samsung Convoy U640 Phone for Verizon Wireless...	Samsung	79.95	5	Passes every drop test onto porcelain tile!	0.0
413838	Samsung Convoy U640 Phone for Verizon Wireless...	Samsung	79.95	3	I returned it because it did not meet my needs...	0.0
413839	Samsung Convoy U640 Phone for Verizon Wireless...	Samsung	79.95	4	Only downside is that apparently Verizon no lo...	0.0

413840 rows × 6 columns

Drop all nans

Removing all rows from the data frame containing missing values.

In [5]:
```
data=data.dropna()
```

In [6]:
```
data_proc = data.copy()
```

In [7]:
```python
data_proc = data_proc.loc[data_proc["Rating"]!=3]
```

In [8]:
```python
data_proc[data_proc["Rating"]==3]
```

Out[8]:

Product Name	Brand Name	Price	Rating	Reviews	Review Votes

In [9]:
```python
def transform(x):
    if x <= 2:
        return -1
    if x > 2:
        return 1
```

In [10]:
```python
data_proc["Rating"] = data_proc['Rating'].map(transform)
```

In [11]:
```python
data_proc[data_proc['Rating']==1]
```

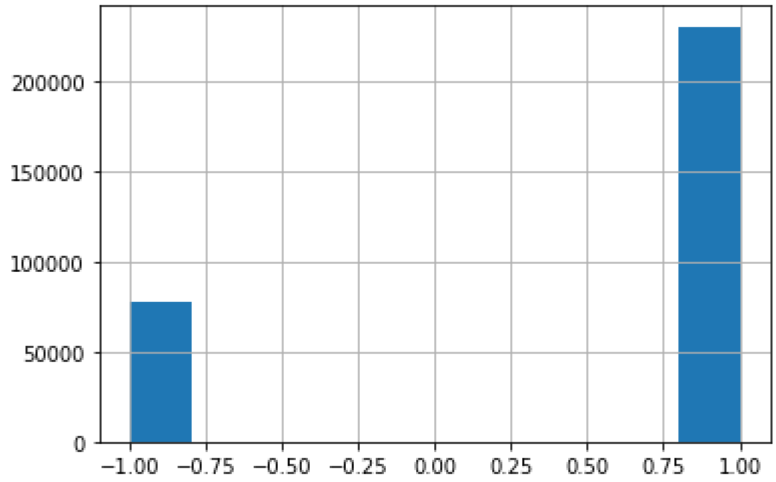
Out[11]:

	Product Name	Brand Name	Price	Rating	Reviews	Review Votes
0	"CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7...	Samsung	199.99	1	I feel so LUCKY to have found this used (phone...	1.0
1	"CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7...	Samsung	199.99	1	nice phone, nice up grade from my pantach revu...	0.0
2	"CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7...	Samsung	199.99	1	Very pleased	0.0
3	"CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7...	Samsung	199.99	1	It works good but it goes slow sometimes but i...	0.0
4	"CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7...	Samsung	199.99	1	Great phone to replace my lost phone. The only...	0.0
...	...	...	...	...	...	...
413830	Samsung Convoy U640 Phone for Verizon Wireless...	Samsung	79.95	1	LOVE IT	0.0
413832	Samsung Convoy U640 Phone for Verizon Wireless...	Samsung	79.95	1	good rugged phone that has a long-lasting batt...	0.0
413835	Samsung Convoy U640 Phone for Verizon Wireless...	Samsung	79.95	1	another great deal great price	0.0
413837	Samsung Convoy U640 Phone for Verizon Wireless...	Samsung	79.95	1	Passes every drop test onto porcelain tile!	0.0
413839	Samsung Convoy U640 Phone for Verizon Wireless...	Samsung	79.95	1	Only downside is that apparently Verizon no lo...	0.0

230674 rows × 6 columns

In [12]:
```python
data_proc["Rating"].hist()
```

Out[12]: <AxesSubplot:>



Go back

II. Balancing class distribution

Imbalance Dataset (Undersampling)

In [13]:
```
# Shuffle the Dataset.
shuffled_df = data_proc.sample(frac=1,random_state=4)
```

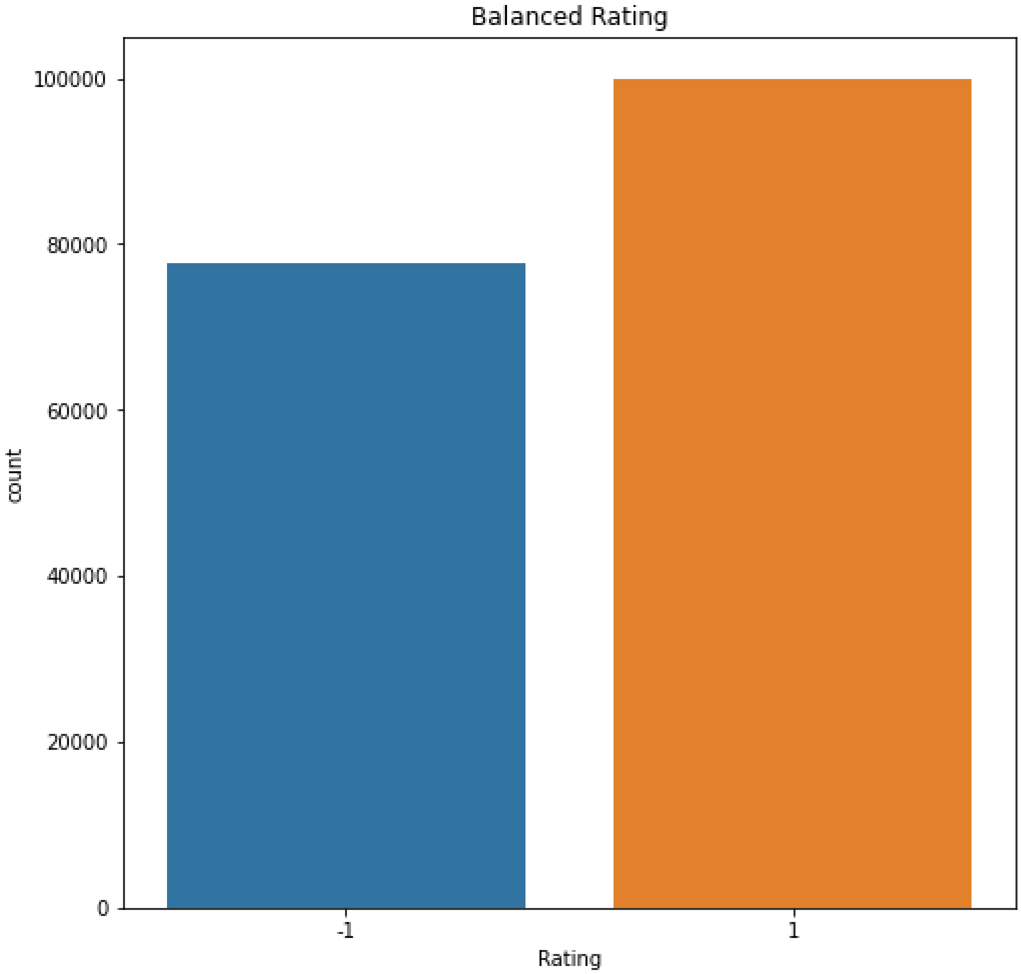
In [14]:
```
# Put all the fraud class in a separate dataset.
negative_review_df = shuffled_df.loc[shuffled_df['Rating'] == -1]
```

In [15]:
```
#Randomly select 492 observations from the non-fraud (majority class)
positive_review_df = shuffled_df.loc[shuffled_df['Rating'] == 1].sample(n=100000,random
```

In [16]:
```
# Concatenate both dataframes again
data_proc_balanced = pd.concat([negative_review_df, positive_review_df])
```

In [17]:
```
#plot the dataset after the undersampling
plt.figure(figsize=(8, 8))
sns.countplot('Rating', data=data_proc_balanced)
plt.title('Balanced Rating')
plt.show()
```

```
C:\Users\uib47087\Anaconda3\envs\mineria\lib\site-packages\seaborn\_decorators.py:43: Fu
tureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the onl
y valid positional argument will be `data`, and passing other arguments without an expli
cit keyword will result in an error or misinterpretation.
  FutureWarning
```

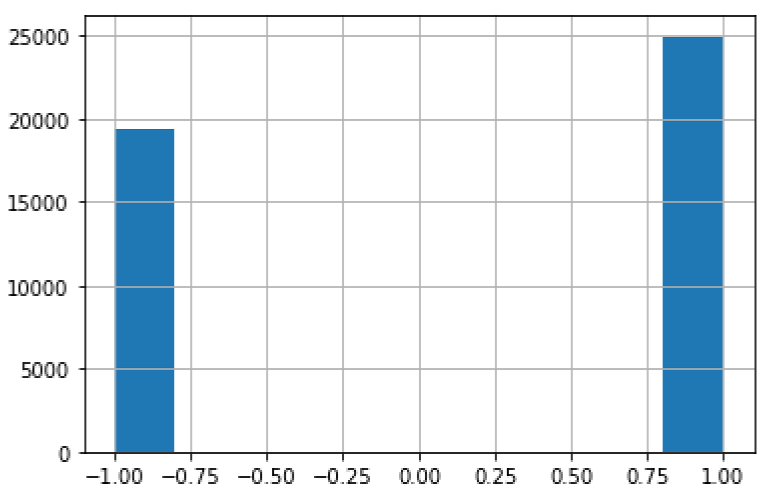


Getting ready for SVM analysis

In [18]:
```
data_to_analyze = data_proc_balanced.sample(frac=0.25,random_state=4)
```

In [19]:
```
data_to_analyze["Rating"].hist()
```

Out[19]: <AxesSubplot:>



Go back

III. Vectorization

Using TfidfVectorizer

In [20]:
```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.feature_selection import SelectPercentile, f_classif, chi2
```

In [21]:
```
vectorizer = TfidfVectorizer(sublinear_tf= True, max_df=0.5, stop_words="english")
```

In [22]:
```
documents = np.array(data_to_analyze["Reviews"])
```

In [23]:
```
vectors = vectorizer.fit_transform(documents)
```

In [24]:
```
X = vectors
y = data_to_analyze["Rating"]
```

In [25]:
```
X
```

Out[25]:
```
<44401x26902 sparse matrix of type '<class 'numpy.float64'>'
	with 707178 stored elements in Compressed Sparse Row format>
```

Using Custom TfidfVectorizer

In [26]:
```
import re
import string
from nltk.stem import SnowballStemmer
from nltk.corpus import stopwords
```

In [27]:
```
punct_marks = re.compile('[%s]' % re.escape(string.punctuation))

def remove_punctuation(sentence):
    new_sent = []

    for token in sentence.split(' '):

        # Look for emails or webpages
        web_chrctrs = ('@','http', 'https')
        if not any(character in token for character in web_chrctrs):
            new_token = re.sub(punct_marks, '', token)

        else:
            new_token = token

        if new_token:
            new_sent.append(new_token)

    return new_sent


stemmer = SnowballStemmer('english')
stops = stopwords.words('english')

def sent_norm(sent_):
    sent = remove_punctuation(sent_)
```

```
    norm_sent = [stemmer.stem(word.lower())
                 for word in sent if word not in stops]
    return norm_sent

custom_tfidf_vectorizer = TfidfVectorizer(tokenizer=sent_norm)
```

In [28]:
```
vectors2 = custom_tfidf_vectorizer.fit_transform(documents)
X2 = vectors2
X2
```

Out[28]:
```
<44401x34257 sparse matrix of type '<class 'numpy.float64'>'
	with 807784 stored elements in Compressed Sparse Row format>
```

Using CountVectorizer

In [29]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

In [30]:
```
count_vectorizer = CountVectorizer()
vectors3 = count_vectorizer.fit_transform(documents)
```

In [31]:
```
X3 = vectors3
X3
```

Out[31]:
```
<44401x27206 sparse matrix of type '<class 'numpy.int64'>'
	with 1257794 stored elements in Compressed Sparse Row format>
```

Using Ngrams

In [32]:
```
cv_ngrams_vectorizer = CountVectorizer(ngram_range=(1,2))
vectors4 = cv_ngrams_vectorizer.fit_transform(documents)
```

In [33]:
```
X4 = vectors4
X4
```

Out[33]:
```
<44401x373603 sparse matrix of type '<class 'numpy.int64'>'
	with 2901081 stored elements in Compressed Sparse Row format>
```

Go back

IV. Data Splitting

In [34]:
```
from sklearn.model_selection import train_test_split
```

Using the vectors from the Standard Tfidf

In [35]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_stat
```

Using the vectors from the Custom Tfidf

In [36]:
```
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y, test_size = 0.25, random
```

Using the vectors from the Count Vectorizer

In [37]:
```
X3_train, X3_test, y3_train, y3_test = train_test_split(X3, y, test_size = 0.25, random
```

Using the vectors from the Count Vectorizer (with Ngrams)

In [38]:
```
X4_train, X4_test, y4_train, y4_test = train_test_split(X4, y, test_size = 0.25, random
```

Go back

V. Data Modeling

In [39]:
```
from sklearn.svm import SVC
classifier = SVC(random_state=0, C=1.0, kernel='rbf', gamma='scale') # for non-linear m
classifier.fit(X_train, y_train)
```

Out[39]:
```
SVC(random_state=0)
```

In [40]:
```
classifier2 = SVC(random_state=0, C=1.0, kernel='rbf', gamma='scale') # for non-linear
classifier2.fit(X2_train, y2_train)
```

Out[40]:
```
SVC(random_state=0)
```

In [41]:
```
classifier3 = SVC(random_state=0, C=1.0, kernel='rbf', gamma='scale') # for non-linear
classifier3.fit(X3_train, y3_train)
```

Out[41]:
```
SVC(random_state=0)
```

In [42]:
```
classifier4 = SVC(random_state=0, C=1.0, kernel='rbf', gamma='scale') # for non-linear
classifier4.fit(X4_train, y4_train)
```

Out[42]:
```
SVC(random_state=0)
```

Notes GridSearch VS Bayesian: Recommended: Bayesian. Aplicate Bayesian algorithm. Parameters Optimization(e.g. c, gama, sigma for the kernel used, hyperparameter as well.)

Go back

VI. Classification

In [43]:
```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

In [44]:
```
y_pred
```

Out[44]:
```
array([-1, -1,  1, ...,  1,  1, -1], dtype=int64)
```

In [45]:
```
results = pd.DataFrame(y_test)
results.rename(columns={"Rating":"Rating_test"}, inplace=True)
```

In [46]:
```
results["Rating_predicted"] = y_pred
```

In [47]:
```
results
```

Out[47]:

	Rating_test	Rating_predicted
146313	-1	-1
206579	-1	-1
177671	1	1
109771	1	1
347801	1	1
...	...	...
411953	1	1
226219	1	1
372587	1	1
13617	1	1
111707	-1	-1

11101 rows × 2 columns

In [48]:
```
predicted = len(results[results["Rating_test"] == results["Rating_predicted"]])
print("Predicted acurately: {0} ratio: {1}".format(predicted, predicted / len(results))
```

```
Predicted acurately: 10387 ratio: 0.9356814701378254
```

In [49]:
```
no_predicted = len(results[results["Rating_test"] != results["Rating_predicted"]])
print("Predicted acurately: {0} ratio: {1}".format(no_predicted, no_predicted / len(res
```

```
Predicted acurately: 714 ratio: 0.06431852986217458
```

For the other vectors...

In [50]:
```
# Predicting the Test set results
y2_pred = classifier2.predict(X2_test)
y3_pred = classifier3.predict(X3_test)
y4_pred = classifier4.predict(X4_test)
```

Go back

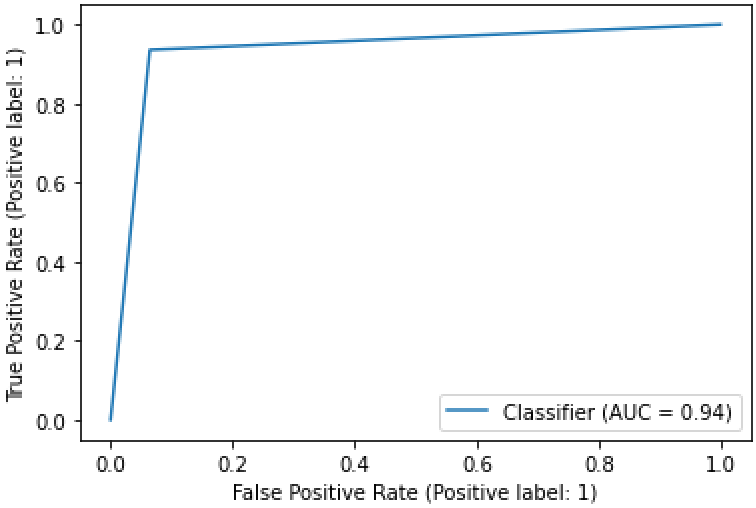
VII. Model Performance

In [51]:
```
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

ROC Curve

In [52]:
```
RocCurveDisplay.from_predictions(y_test, y_pred)
```

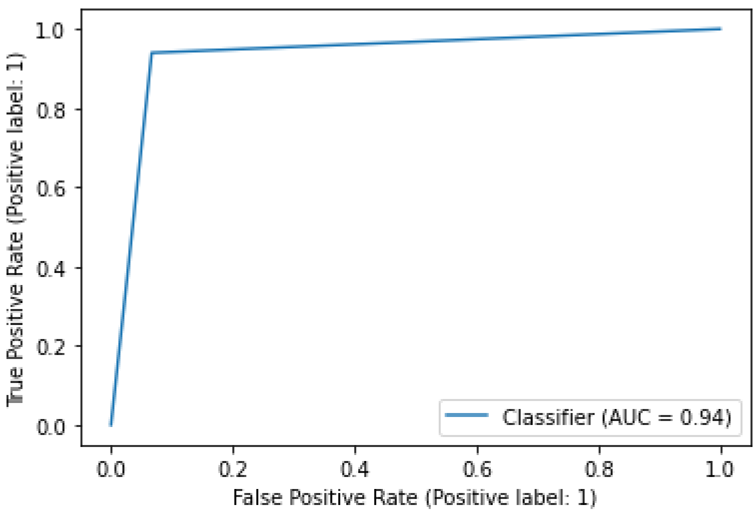
Out[52]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x246520eea48>
```



In [53]:

```
RocCurveDisplay.from_predictions(y2_test, y2_pred)
```

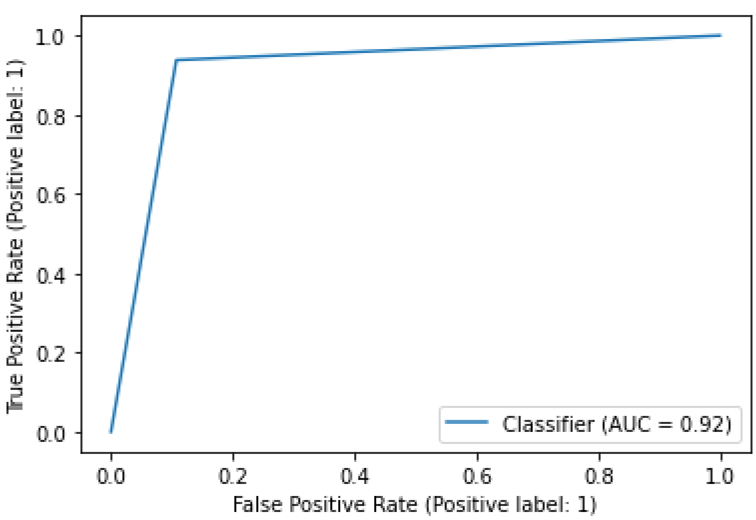
Out[53]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x246520e8648>



In [54]:

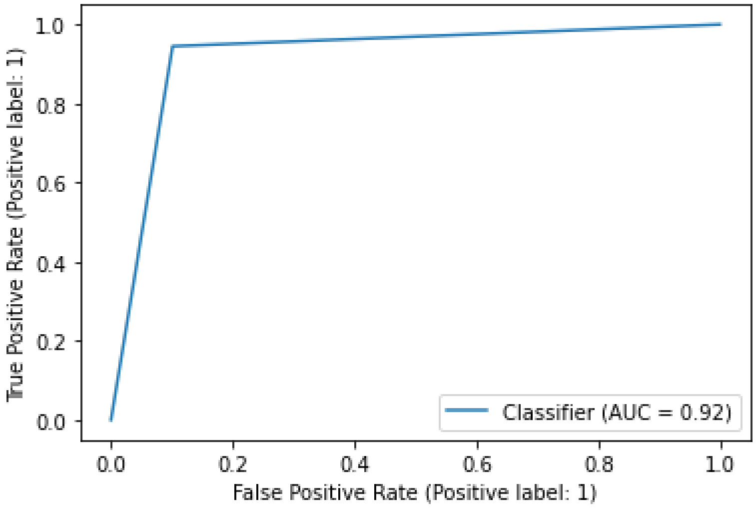
```
RocCurveDisplay.from_predictions(y3_test, y3_pred)
```

Out[54]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x246560e0888>



In [55]:
```
RocCurveDisplay.from_predictions(y4_test, y4_pred)
```

Out[55]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x24656c11648>



AUC

In [56]:
```
auc = roc_auc_score(y_test, y_pred)
auc
```

Out[56]: 0.935660684476294

In [57]:
```
auc2 = roc_auc_score(y2_test, y2_pred)
auc2
```

Out[57]: 0.9363702424839941

In [58]:
```
auc3 = roc_auc_score(y3_test, y3_pred)
auc3
```

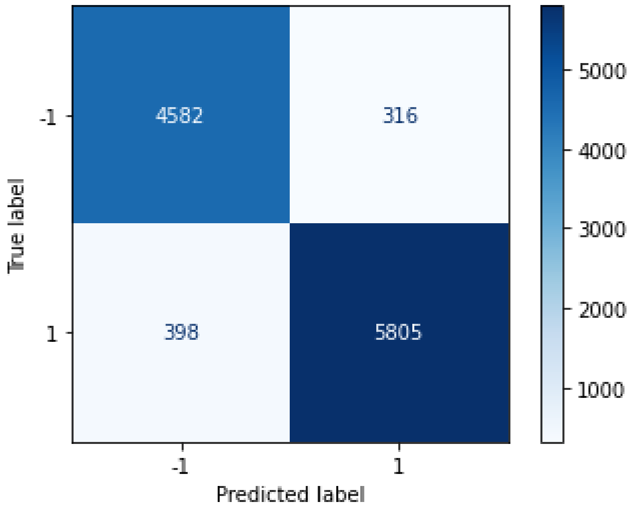
Out[58]: 0.915271243178675

In [59]:
```
auc4 = roc_auc_score(y4_test, y4_pred)
auc4
```

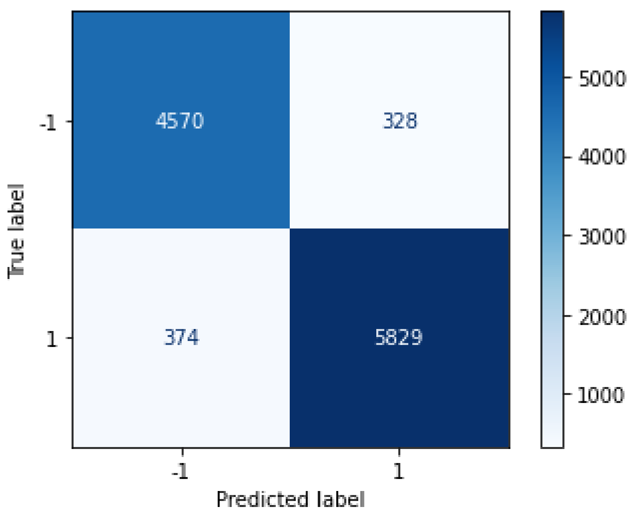
Out[59]: 0.9217997824654056

Confusion Matrix

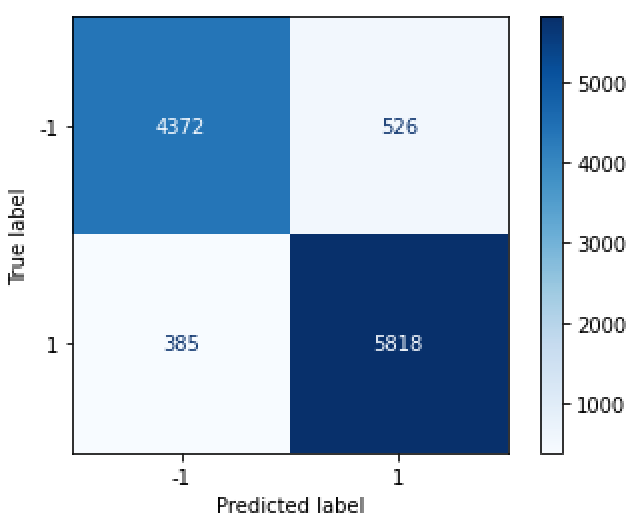
In [60]:
```
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap=plt.cm.Blues)
plt.show()
```



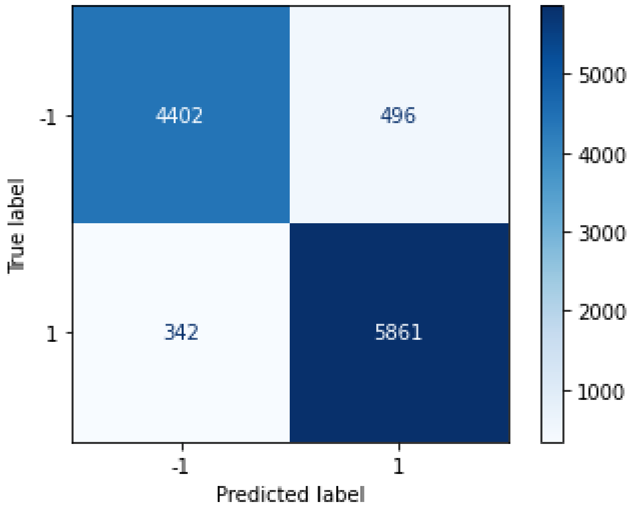
In [61]:
```
ConfusionMatrixDisplay.from_predictions(y2_test, y2_pred, cmap=plt.cm.Blues)
plt.show()
```



In [62]:
```
ConfusionMatrixDisplay.from_predictions(y3_test, y3_pred, cmap=plt.cm.Blues)
plt.show()
```



In [63]:
```
ConfusionMatrixDisplay.from_predictions(y4_test, y4_pred, cmap=plt.cm.Blues)
plt.show()
```



Go back

VIII. Bayesian Optimization

In [64]:
```
import matplotlib.tri as tri

from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from hyperopt import fmin, tpe, Trials, hp, STATUS_OK
```

In [65]:
```
def objective(args):
    '''Define the loss function / objective of our model.

    We will be using an SVM parameterized by the regularization parameter C
    and the parameter gamma.

    The C parameter trades off correct classification of training examples
    against maximization of the decision function's margin. For larger values
    of C, a smaller margin will be accepted.

    The gamma parameter defines how far the influence of a single training
    example reaches, with larger values meaning 'close'.
    '''
    C, gamma = args
    model = SVC(C=10 ** C, gamma=10 ** gamma, random_state=0)
    loss = 1 - cross_val_score(estimator=model, X=X_train, y=y_train, scoring='roc_auc'
    return {'params': {'C': C, 'gamma': gamma}, 'loss': loss, 'status': STATUS_OK }
```

In [66]:
```
def objective2(args):

    C, gamma = args
    model = SVC(C=10 ** C, gamma=10 ** gamma, random_state=0)
    loss = 1 - cross_val_score(estimator=model, X=X2_train, y=y2_train, scoring='roc_au
    return {'params': {'C': C, 'gamma': gamma}, 'loss': loss, 'status': STATUS_OK }
```

In [67]:
```
def objective3(args):

    C, gamma = args
    model = SVC(C=10 ** C, gamma=10 ** gamma, random_state=0)
```

```
    loss = 1 - cross_val_score(estimator=model, X=X3_train, y=y3_train, scoring='roc_au
    return {'params': {'C': C, 'gamma': gamma}, 'loss': loss, 'status': STATUS_OK }
```

In [68]:
```
def objective4(args):

    C, gamma = args
    model = SVC(C=10 ** C, gamma=10 ** gamma, random_state=0)
    loss = 1 - cross_val_score(estimator=model, X=X4_train, y=y4_train, scoring='roc_au
    return {'params': {'C': C, 'gamma': gamma}, 'loss': loss, 'status': STATUS_OK }
```

In [ ]:
```
trials = Trials()
best_m1 = fmin(objective,
    space=[hp.uniform('C', -4., 1.), hp.uniform('gamma', -4., 1.)],
    algo=tpe.suggest,
    max_evals=100,
    trials=trials)
```

In [ ]:
```
trials2 = Trials()
best_m2 = fmin(objective2,
    space=[hp.uniform('C', -4., 1.), hp.uniform('gamma', -4., 1.)],
    algo=tpe.suggest,
    max_evals=100,
    trials=trials2)
```

In [ ]:
```
trials3 = Trials()
best_m3 = fmin(objective3,
    space=[hp.uniform('C', -4., 1.), hp.uniform('gamma', -4., 1.)],
    algo=tpe.suggest,
    max_evals=100,
    trials=trials3)
```

In [ ]:
```
trials4 = Trials()
best_m4 = fmin(objective4,
    space=[hp.uniform('C', -4., 1.), hp.uniform('gamma', -4., 1.)],
    algo=tpe.suggest,
    max_evals=100,
    trials=trials4)
```

In [ ]:
```
print(best_m1, best_m2, best_m3, best_m4)
```

In [ ]:
```
results = trials.results
ar = np.zeros(shape=(1000,3))
for i, r in enumerate(results):
    C = r['params']['C']
    gamma = r['params']['gamma']
    loss = r['loss']
    ar[i] = C, gamma, loss
```

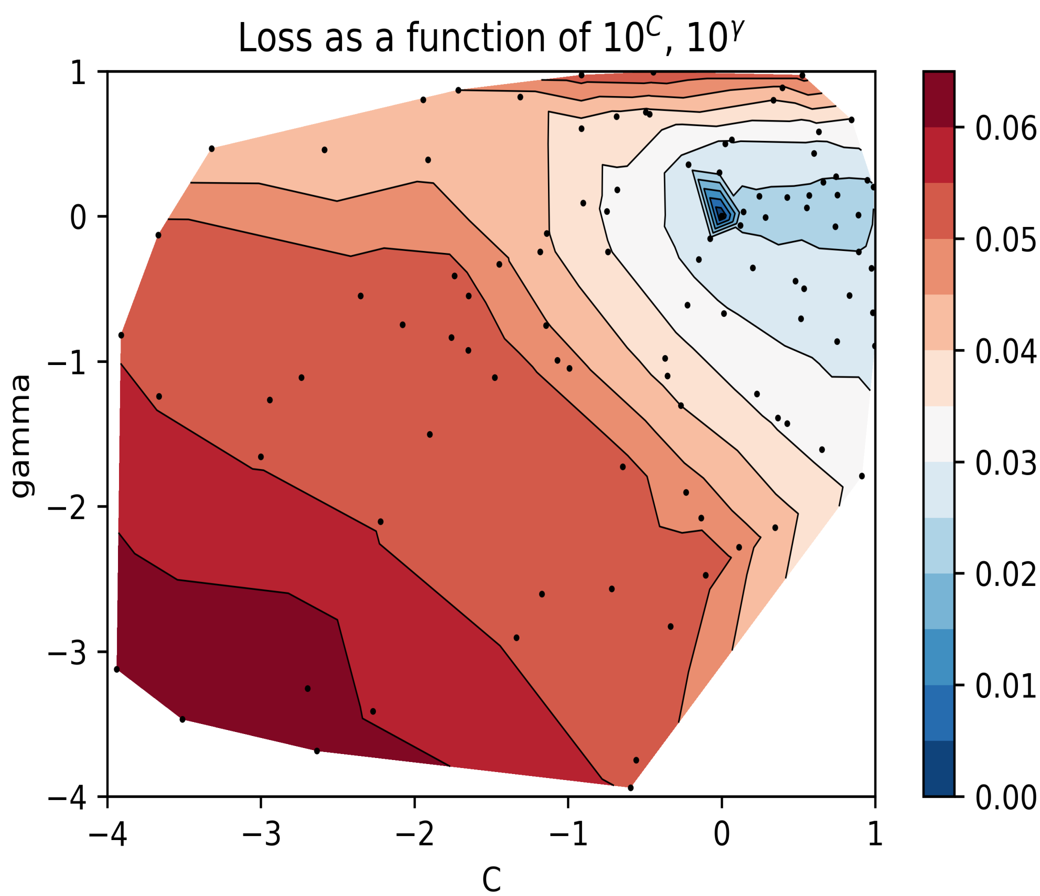
In [ ]:
```
C, gamma, loss = ar[:, 0], ar[:, 1], ar[:, 2]

fig, ax = plt.subplots(nrows=1)
ax.tricontour(C, gamma, loss, levels=14, linewidths=0.5, colors='k')
cntr = ax.tricontourf(C, gamma, loss, levels=14, cmap="RdBu_r")

fig.colorbar(cntr, ax=ax)
ax.plot(C, gamma, 'ko', ms=1)
ax.set(xlim=(-4, 1), ylim=(-4, 1))
```

```
plt.title('Model I - Loss as a function of $10^C$, $10^\gamma$')
plt.xlabel('C')
plt.ylabel('gamma')

plt.show()
```

Optimization results 1

TF-IDF Vectorizer with no data normalization

- Hyperparameter optimal values
  - **C**: 0.568940290521819,
  - **gamma** : 0.14524519347606762



In [ ]:
```
results2 = trials2.results
ar2 = np.zeros(shape=(1000,3))
for i, r in enumerate(results2):
    C2 = r['params']['C']
    gamma2 = r['params']['gamma']
    loss2 = r['loss']
    ar2[i] = C2, gamma2, loss2
```

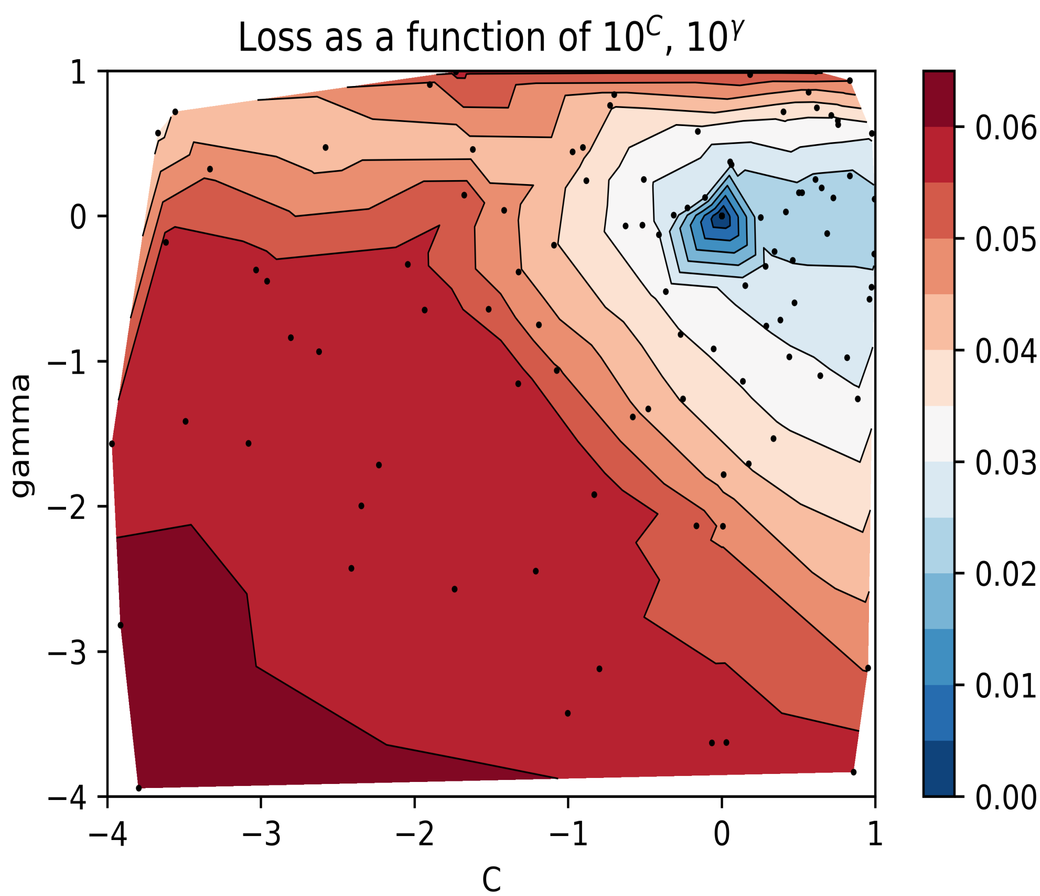
In [ ]:
```
C, gamma, loss = ar2[:, 0], ar2[:, 1], ar2[:, 2]

fig, ax = plt.subplots(nrows=1)
```

```
ax.tricontour(C, gamma, loss, levels=14, linewidths=0.5, colors='k')
cntr = ax.tricontourf(C, gamma, loss, levels=14, cmap="RdBu_r")

fig.colorbar(cntr, ax=ax)
ax.plot(C, gamma, 'ko', ms=1)
ax.set(xlim=(-4, 1), ylim=(-4, 1))
plt.title('Model II - Loss as a function of $10^C$, $10^\gamma$')
plt.xlabel('C')
plt.ylabel('gamma')

plt.show()
```

Optimization results 2

TF-IDF Vectorizer with data normalization

- Hyperparameter optimal values
  - C: 0.4169145993204806,
  - gamma': 0.02922866354035983



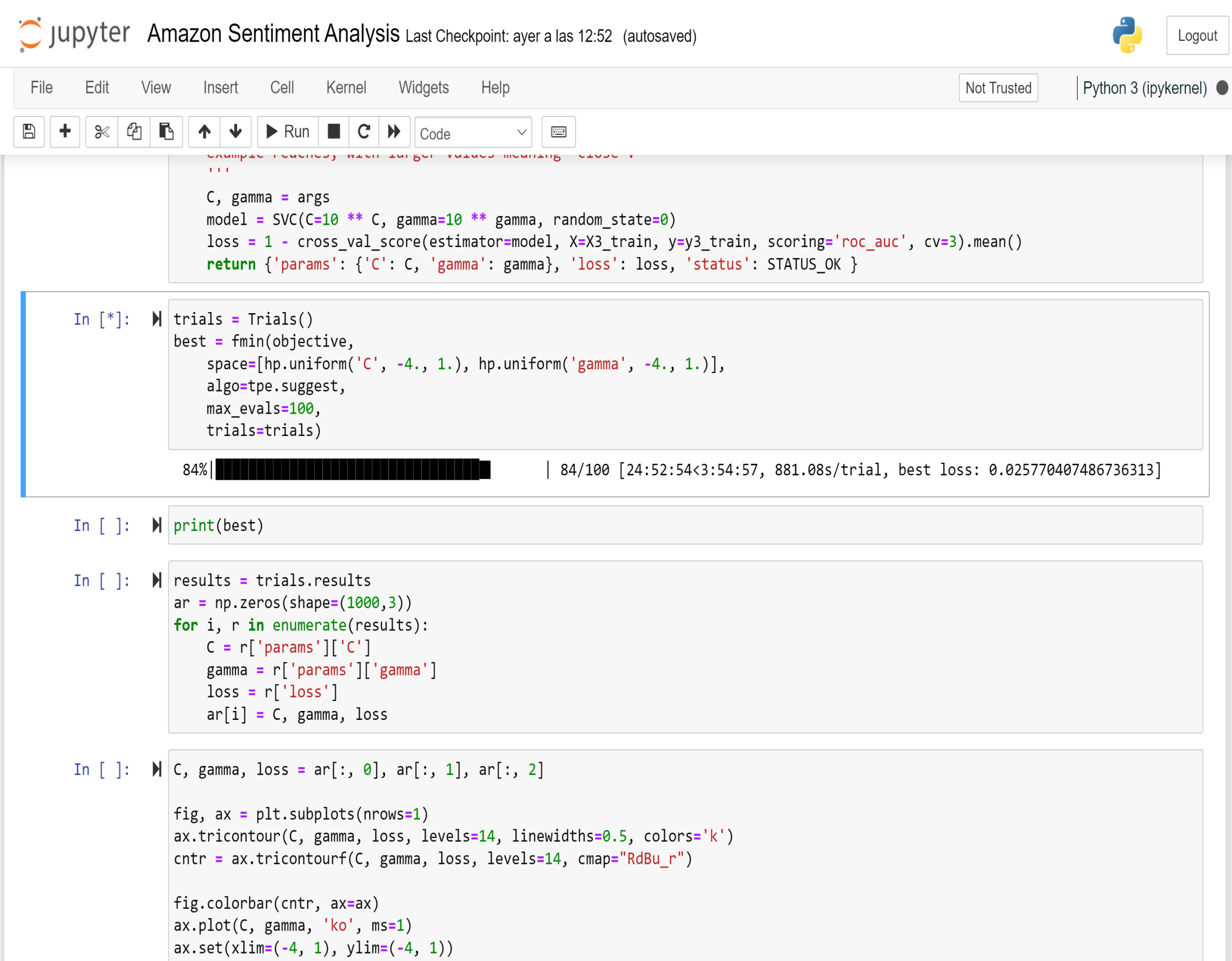
In [ ]:
```
results3 = trials3.results
ar3 = np.zeros(shape=(1000,3))
for i, r in enumerate(results3):
    C3 = r['params']['C']
    gamma3 = r['params']['gamma']
```

```
    loss3 = r['loss']
    ar3[i] = C3, gamma3, loss3
```

In [ ]:
```
C, gamma, loss = ar3[:, 0], ar3[:, 1], ar3[:, 2]

fig, ax = plt.subplots(nrows=1)
ax.tricontour(C, gamma, loss, levels=14, linewidths=0.5, colors='k')
cntr = ax.tricontourf(C, gamma, loss, levels=14, cmap="RdBu_r")

fig.colorbar(cntr, ax=ax)
ax.plot(C, gamma, 'ko', ms=1)
ax.set(xlim=(-4, 1), ylim=(-4, 1))
plt.title('Model III - Loss as a function of $10^C$, $10^\gamma$')
plt.xlabel('C')
plt.ylabel('gamma')

plt.show()
```

Count Vectorizer (Current Status)

Hardware limitations are making no possible for now to deliver final optimization results. Please refer to the image below to corroborate process status.



In [ ]:
```
results4 = trials4.results
ar4 = np.zeros(shape=(1000,3))
for i, r in enumerate(results4):
    C4 = r['params']['C']
```

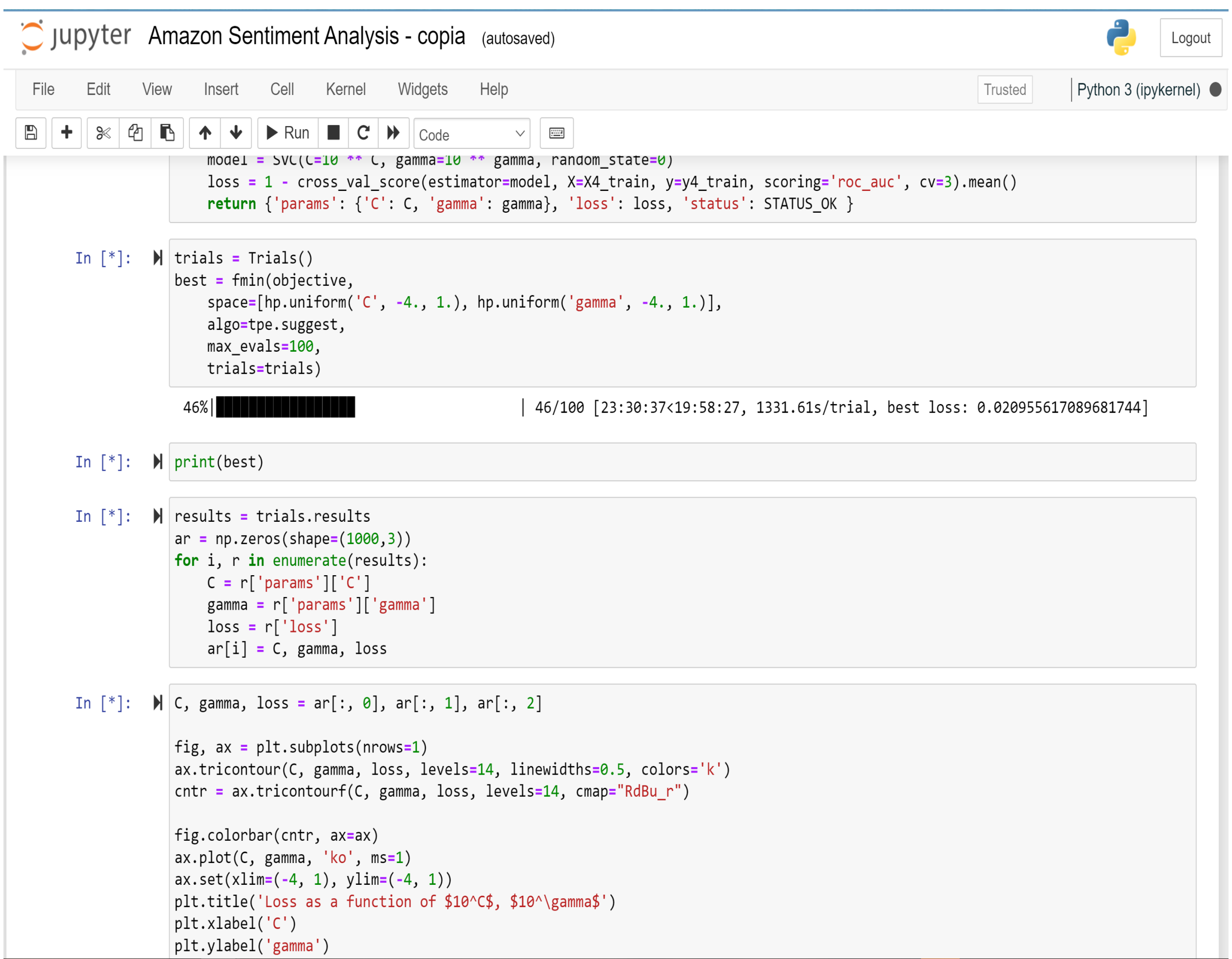
```
    gamma4 = r['params']['gamma']
    loss4 = r['loss']
    ar4[i] = C4, gamma4, loss4
```

In [ ]:
```
C, gamma, loss = ar4[:, 0], ar4[:, 1], ar4[:, 2]

fig, ax = plt.subplots(nrows=1)
ax.tricontour(C, gamma, loss, levels=14, linewidths=0.5, colors='k')
cntr = ax.tricontourf(C, gamma, loss, levels=14, cmap="RdBu_r")

fig.colorbar(cntr, ax=ax)
ax.plot(C, gamma, 'ko', ms=1)
ax.set(xlim=(-4, 1), ylim=(-4, 1))
plt.title('Model IV - Loss as a function of $10^C$, $10^\gamma$')
plt.xlabel('C')
plt.ylabel('gamma')

plt.show()
```

NGrams (Current Status)

Due to vector size, it was not possible to deliver results for this optimization by the time this report will be delivered. Current progress is less than 50 percent so far.

best loss .02095



jupyter Amazon Sentiment Analysis - copia (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

```
    model = SVC(C=10 ** C, gamma=10 ** gamma, random_state=0)
    loss = 1 - cross_val_score(estimator=model, X=X4_train, y=y4_train, scoring='roc_auc', cv=3).mean()
    return {'params': {'C': C, 'gamma': gamma}, 'loss': loss, 'status': STATUS_OK }
```

In [*]:
```
trials = Trials()
best = fmin(objective,
    space=[hp.uniform('C', -4., 1.), hp.uniform('gamma', -4., 1.)],
    algo=tpe.suggest,
    max_evals=100,
    trials=trials)
```

```
46%|██████████████          | 46/100 [23:30:37<19:58:27, 1331.61s/trial, best loss: 0.020955617089681744]
```

In [*]:
```
print(best)
```

In [*]:
```
results = trials.results
ar = np.zeros(shape=(1000,3))
for i, r in enumerate(results):
    C = r['params']['C']
    gamma = r['params']['gamma']
    loss = r['loss']
    ar[i] = C, gamma, loss
```

In [*]:
```
C, gamma, loss = ar[:, 0], ar[:, 1], ar[:, 2]

fig, ax = plt.subplots(nrows=1)
ax.tricontour(C, gamma, loss, levels=14, linewidths=0.5, colors='k')
cntr = ax.tricontourf(C, gamma, loss, levels=14, cmap="RdBu_r")

fig.colorbar(cntr, ax=ax)
ax.plot(C, gamma, 'ko', ms=1)
ax.set(xlim=(-4, 1), ylim=(-4, 1))
plt.title('Loss as a function of $10^C$, $10^\gamma$')
plt.xlabel('C')
plt.ylabel('gamma')
```

Go back

IX. Conclusions

Some challenges were faced in the initial stages of this project:

1. Unbalanced dataset
2. Translation from text to numeric values through vectorization techniques (TF-IDF Vectorization)
3. Computational challenges were a factor for the optimziation workload. Mainly with the ngrams run. Results still pending to conclude until this time.

Though, team was not able to complete full optimization for all the cases (CountVectorizer and NGrams) we learned:

1. Using different techniques for vectorization we were able to benchmark the performance of the models and observe that the chosen vectorization technique can impact on the performance metrics.

2. Considerable improvement was observed with the normalized optimization. (In the graph more values fell into the blue zone, which implies smaller parameters)

3. It was common across all the optimization models a best value around 0.024 or less.

Go back

8 References

1. A Gentle Introduction to the Bag-of-Words Model

2. Support Vector Machines

3. Scikit-Optimize for Hyperparameter Tuning in Machine Learning

4. How to Implement Bayesian Optimization from Scratch in Python

5. 11 Important Model Evaluation Metrics for Machine Learning Everyone should know

6. A Guide to Text Classification and Sentiment Analysis

7. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning

8. Bayesian optimization for hyperparameter tuning