

## Assignment 1

**1. (non-probabilistic) k-NN classifier**

a. Choosing the 'Large movie review dataset' because due to a large number of datapoints  $p(x,y) / D(x,y)$  will be derived properly.

b. For this feature map, we use

```
# Remove all the special characters
# remove all single characters
# Remove single characters from the start
# Substituting multiple spaces with single space
# Removing prefixed 'b'
# Converting to Lowercase
```

This presents the more processed data for the classifier to map with output.

c. For 70% train and 30% test data division

**i. For k=1 and p = 1 :**

	precision	recall	f1-score	support
neg	0.52	0.96	0.67	3787
pos	0.69	0.09	0.16	3713
micro avg	0.53	0.53	0.53	7500
macro avg	0.61	0.53	0.42	7500
weighted avg	0.60	0.53	0.42	7500

Confusion matrix

```
[ [3636    151]
  [3373    340]]
```

Accuracy score = 0.5301333333333333

**ii. For k = 1 and p = 2 :**

Confusion matrix

```
[[2490 1297]
 [1019 2694]]
```

	precision	recall	f1-score	support
neg	0.71	0.66	0.68	3787
pos	0.68	0.73	0.70	3713
micro avg	0.69	0.69	0.69	7500
macro avg	0.69	0.69	0.69	7500
weighted avg	0.69	0.69	0.69	7500

Accuracy score = 0.6912

## iii. For k=1 and p=inf :

Confusion matrix:

[[2396 1402]

[1561 2123]]

	Precision	recall	f1-score	support
neg	0.61	0.63	0.62	3798
pos	0.60	0.58	0.59	3684
micro avg	0.60	0.60	0.60	7482
macro avg	0.60	0.60	0.60	7482
weighted avg	0.60	0.60	0.60	7482

Accuracy score = 0.6039828922747928

## iv. For k=3 and p = 1 :

Confusion matrix:

[[3760 27]

[3600 113]]

	Precision	recall	f1-score	support
neg	0.51	0.99	0.67	3787
pos	0.81	0.03	0.06	3713
micro avg	0.52	0.52	0.52	7500
macro avg	0.66	0.51	0.37	7500
weighted avg	0.66	0.52	0.37	7500

Accuracy score = 0.5164

## v. For k=3 and p=2 :

Confusion matrix:

[[2558 1229]

[ 915 2798]]

	Precision	recall	f1-score	support
neg	0.74	0.68	0.70	3787
pos	0.69	0.75	0.72	3713
micro avg	0.71	0.71	0.71	7500
macro avg	0.72	0.71	0.71	7500
weighted avg	0.72	0.71	0.71	7500

Accurecy score = 0.7141333333333333

**vi. For k=3 and p =inf :**

Confusion matrix

[[2440 1347]

[1535 2178]]

Precision recall f1-score support

neg	0.61	0.64	0.63	3787
pos	0.62	0.59	0.60	3713
micro avg	0.62	0.62	0.62	7500
macro avg	0.62	0.62	0.62	7500
weighted avg	0.62	0.62	0.62	7500

Accuracy score =0.6157333333333334

**vii. For k = 5 and p = 1 :**

Confusion matrix

[[3779 8]

[3644 69]]

Precision recall f1-score support

neg	0.51	1.00	0.67	3787
pos	0.90	0.02	0.04	3713
micro avg	0.51	0.51	0.51	7500
macro avg	0.70	0.51	0.36	7500
weighted avg	0.70	0.51	0.36	7500

Accuracy score = 0.5130666666666667

**viii. For k = 5 and p = 2 :**

Confusion matrix :

[[2604 1183]

[840 2873]]

Precision recall f1-score support

neg	0.76	0.69	0.72	3787
pos	0.71	0.77	0.74	3713
micro avg	0.73	0.73	0.73	7500
macro avg	0.73	0.73	0.73	7500
weighted avg	0.73	0.73	0.73	7500

Accuracy score = 0.7302666666666666

ix. For k = 5 and p = inf :

Confusion matrix :

[[2514 1273]

[1551 2162]]

Precision recall f1-score support

neg	0.62	0.66	0.64	3787
pos	0.63	0.58	0.60	3713
micro avg	0.62	0.62	0.62	7500
macro avg	0.62	0.62	0.62	7500
weighted avg	0.62	0.62	0.62	7500

Accuracy score = 0.6234666666666666

### 3. KDE based Bayes classifier :

Kernel density estimation uses kernel functions for each feature summing up gives a more accurate density function than histograms.

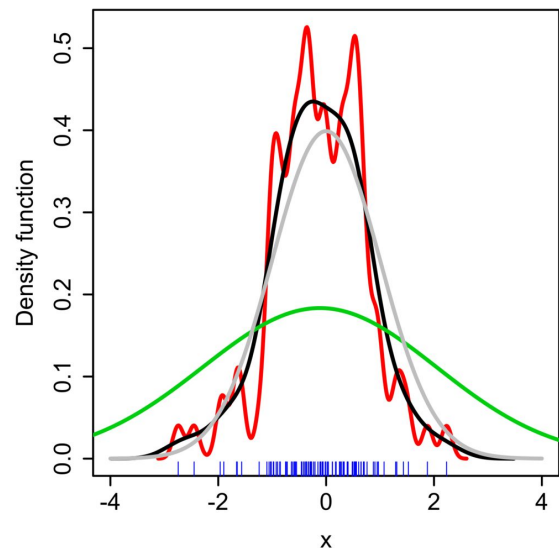
$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^n K_h(x - x_i) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right)$$

K is the kernel function and 'h' is bandwidth which defines how sharp or smooth the distribution can be.

Denser the data set nerby more will be the probability of occurrence.

The below data is plotted with a gaussian kernel with h = 0.337, 0.5, 2.

\*(the code I was running had many bugs and i wasn't able to fix it. So, pardon me for that sir.)



### 3. kernelized SVM

We did the data preprocessing as above.

In kernelized SVM we used the 'RBF (radial basis function)' kernel for classification.

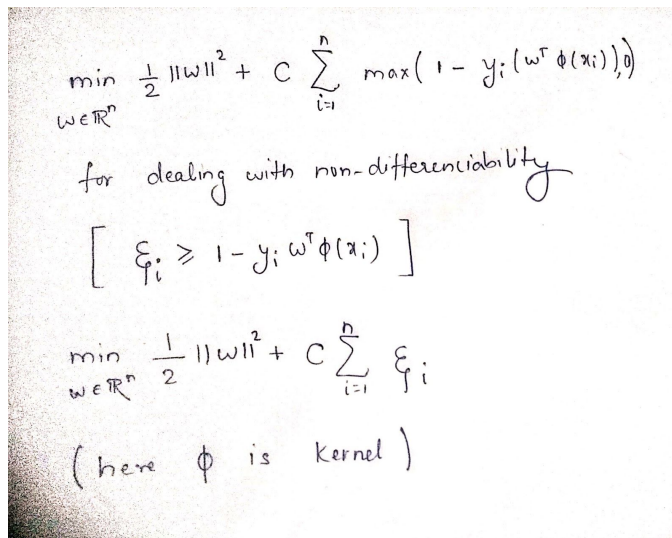
For 50% train and 50% test data division

The RBF kernel on two samples  $\mathbf{x}$  and  $\mathbf{x}'$ , represented as feature vectors in some *input space*, is defined as

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

Where  $\gamma = 1/(2\sigma^2)$

For loss function in kernelized SVM



Handwritten mathematical derivation of the SVM loss function with slack variables:

$$\min_{\mathbf{w} \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \max(1 - y_i (\mathbf{w}^T \phi(\mathbf{x}_i)), 0)$$

for dealing with non-differentiability

$$\left[ \xi_i \geq 1 - y_i \mathbf{w}^T \phi(\mathbf{x}_i) \right]$$

$$\min_{\mathbf{w} \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i$$

(here  $\phi$  is kernel)

[[3528 2752]

[ 301 5888]]

	precision	recall	f1-score	support
neg	0.92	0.56	0.70	6280
pos	0.68	0.95	0.79	6189

accuracy		0.76	12469
macro avg	0.80	0.76	0.75 12469
weighted avg	0.80	0.76	0.75 12469

0.7551527788916513

[[3528 2752]

[ 301 5888]]

precision recall f1-score support

neg 0.92 0.56 0.70 6280

pos 0.68 0.95 0.79 6189

accuracy 0.76 12469

macro avg 0.80 0.76 0.75 12469

weighted avg 0.80 0.76 0.75 12469

0.7551527788916513

[[3528 2752]

[ 301 5888]]

precision recall f1-score support

neg 0.92 0.56 0.70 6280

pos 0.68 0.95 0.79 6189

accuracy 0.76 12469

macro avg 0.80 0.76 0.75 12469

weighted avg 0.80 0.76 0.75 12469

0.7551527788916513

[[3528 2752]

[ 301 5888]]

precision recall f1-score support

neg 0.92 0.56 0.70 6280

pos 0.68 0.95 0.79 6189

accuracy 0.76 12469

macro avg 0.80 0.76 0.75 12469

weighted avg 0.80 0.76 0.75 12469

0.7551527788916513

[[3528 2752]

[ 301 5888]]

precision recall f1-score support

neg 0.92 0.56 0.70 6280

pos 0.68 0.95 0.79 6189

accuracy 0.76 12469

macro avg	0.80	0.76	0.75	12469
weighted avg	0.80	0.76	0.75	12469

0.7551527788916513

[[5303 977]

[ 613 5576]]

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

neg	0.90	0.84	0.87	6280
-----	------	------	------	------

pos	0.85	0.90	0.88	6189
-----	------	------	------	------

accuracy			0.87	12469
macro avg	0.87	0.87	0.87	12469
weighted avg	0.87	0.87	0.87	12469

0.8724837597241158

[[5303 977]

[ 613 5576]]

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

neg	0.90	0.84	0.87	6280
-----	------	------	------	------

pos	0.85	0.90	0.88	6189
-----	------	------	------	------

accuracy			0.87	12469
macro avg	0.87	0.87	0.87	12469
weighted avg	0.87	0.87	0.87	12469

0.8724837597241158

[[5303 977]

[ 613 5576]]

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

neg	0.90	0.84	0.87	6280
-----	------	------	------	------

pos	0.85	0.90	0.88	6189
-----	------	------	------	------

accuracy			0.87	12469
macro avg	0.87	0.87	0.87	12469
weighted avg	0.87	0.87	0.87	12469

0.8724837597241158

[[5303 977]

[ 613 5576]]

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

neg	0.90	0.84	0.87	6280
pos	0.85	0.90	0.88	6189

accuracy			0.87	12469
macro avg	0.87	0.87	0.87	12469
weighted avg	0.87	0.87	0.87	12469

0.8724837597241158

[[5303 977]

[ 613 5576]]

precision recall f1-score support

neg	0.90	0.84	0.87	6280
pos	0.85	0.90	0.88	6189

accuracy			0.87	12469
macro avg	0.87	0.87	0.87	12469
weighted avg	0.87	0.87	0.87	12469

0.8724837597241158

[[5394 886]

[ 735 5454]]

precision recall f1-score support

neg	0.88	0.86	0.87	6280
pos	0.86	0.88	0.87	6189

accuracy			0.87	12469
macro avg	0.87	0.87	0.87	12469
weighted avg	0.87	0.87	0.87	12469

0.8699975940332023

[[5394 886]

[ 735 5454]]

precision recall f1-score support

neg	0.88	0.86	0.87	6280
pos	0.86	0.88	0.87	6189

accuracy			0.87	12469
macro avg	0.87	0.87	0.87	12469
weighted avg	0.87	0.87	0.87	12469



0.8699975940332023

[[5394 886]

[ 735 5454]]

precision recall f1-score support

neg 0.88 0.86 0.87 6280

pos 0.86 0.88 0.87 6189

accuracy 0.87 12469

macro avg 0.87 0.87 0.87 12469

weighted avg 0.87 0.87 0.87 12469

0.8699975940332023

[[5394 886]

[ 735 5454]]

precision recall f1-score support

neg 0.88 0.86 0.87 6280

pos 0.86 0.88 0.87 6189

accuracy 0.87 12469

macro avg 0.87 0.87 0.87 12469

weighted avg 0.87 0.87 0.87 12469

0.8699975940332023

[[5394 886]

[ 735 5454]]

precision recall f1-score support

neg 0.88 0.86 0.87 6280

pos 0.86 0.88 0.87 6189

accuracy 0.87 12469

macro avg 0.87 0.87 0.87 12469

weighted avg 0.87 0.87 0.87 12469

0.8699975940332023

[[5315 965]

[ 896 5293]]

precision recall f1-score support

neg 0.86 0.85 0.85 6280

pos	0.85	0.86	0.85	6189
accuracy			0.85	12469
macro avg	0.85	0.85	0.85	12469
weighted avg	0.85	0.85	0.85	12469

0.8507498596519368

[[5315 965]

[ 896 5293]]

precision recall f1-score support

neg	0.86	0.85	0.85	6280
pos	0.85	0.86	0.85	6189

accuracy			0.85	12469
macro avg	0.85	0.85	0.85	12469
weighted avg	0.85	0.85	0.85	12469

0.8507498596519368

[[5315 965]

[ 896 5293]]

precision recall f1-score support

neg	0.86	0.85	0.85	6280
pos	0.85	0.86	0.85	6189

accuracy			0.85	12469
macro avg	0.85	0.85	0.85	12469
weighted avg	0.85	0.85	0.85	12469

0.8507498596519368

[[5315 965]

[ 896 5293]]

precision recall f1-score support

neg	0.86	0.85	0.85	6280
pos	0.85	0.86	0.85	6189

accuracy			0.85	12469
macro avg	0.85	0.85	0.85	12469
weighted avg	0.85	0.85	0.85	12469

0.8507498596519368

[[5315 965]

[ 896 5293]]

precision recall f1-score support

neg 0.86 0.85 0.85 6280

pos 0.85 0.86 0.85 6189

accuracy 0.85 12469

macro avg 0.85 0.85 0.85 12469

weighted avg 0.85 0.85 0.85 12469

0.8507498596519368

[[5218 1062]

[1070 5119]]

precision recall f1-score support

neg 0.83 0.83 0.83 6280

pos 0.83 0.83 0.83 6189

accuracy 0.83 12469

macro avg 0.83 0.83 0.83 12469

weighted avg 0.83 0.83 0.83 12469

0.8290159595797578

[[5218 1062]

[1070 5119]]

precision recall f1-score support

neg 0.83 0.83 0.83 6280

pos 0.83 0.83 0.83 6189

accuracy 0.83 12469

macro avg 0.83 0.83 0.83 12469

weighted avg 0.83 0.83 0.83 12469

0.8290159595797578

[[5218 1062]

[1070 5119]]

precision recall f1-score support

neg 0.83 0.83 0.83 6280

pos 0.83 0.83 0.83 6189

accuracy		0.83	12469
macro avg	0.83	0.83	0.83 12469
weighted avg	0.83	0.83	0.83 12469

0.8290159595797578

[[5218 1062]

[1070 5119]]

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

neg	0.83	0.83	0.83	6280
pos	0.83	0.83	0.83	6189

accuracy		0.83	12469
macro avg	0.83	0.83	0.83 12469
weighted avg	0.83	0.83	0.83 12469

0.8290159595797578

[[5218 1062]

[1070 5119]]

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

neg	0.83	0.83	0.83	6280
pos	0.83	0.83	0.83	6189

accuracy		0.83	12469
macro avg	0.83	0.83	0.83 12469
weighted avg	0.83	0.83	0.83 12469

0.8290159595797578

#### 4. Logistic regression

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^n \log(\exp(-y_i(X_i^T w + c)) + 1).$$

As stated above the loss function for logistic regression where C is the hyperparameter by tuning that we can get the desired classifiers. (we applied same feature extraction as above)

##### i. For C = 0.001

Confusion matrix :

[[4544 1736]

[ 722 5467]]

	Precision	recall	f1-score	support
neg	0.86	0.72	0.79	6280
pos	0.76	0.88	0.82	6189
accuracy			0.80	12469
macro avg	0.81	0.80	0.80	12469
weighted avg	0.81	0.80	0.80	12469

Accuracy Score = 0.8028711203785388

##### ii. For C = 0.01

Confusion matrix :

[[5008 1272]

[ 782 5407]]

	Precision	recall	f1-score	support
neg	0.86	0.80	0.83	6280
pos	0.81	0.87	0.84	6189
accuracy			0.84	12469
macro avg	0.84	0.84	0.84	12469
weighted avg	0.84	0.84	0.84	12469

Accuracy score = 0.8352714732536691

##### iii. For C = 0.1

Confusion matrix : [[5343 937]

[ 708 5481]]

	precision	recall	f1-score	support
neg	0.88	0.85	0.87	6280
pos	0.85	0.89	0.87	6189
accuracy			0.87	12469
macro avg	0.87	0.87	0.87	12469
weighted avg	0.87	0.87	0.87	12469

Accuracy score = 0.8680728205950757

**iv. For C = 1**

Confusion matrix :

[[5453 827]

[ 723 5466]]

	Precision	recall	f1-score	support
neg	0.88	0.87	0.88	6280
pos	0.87	0.88	0.88	6189
accuracy			0.88	12469
macro avg	0.88	0.88	0.88	12469
weighted avg	0.88	0.88	0.88	12469

Accuracy score = 0.8756917154543268

**v. For C = 10**

Confusion matrix: [[5378 902]

[ 836 5353]]

	Precision	recall	f1-score	support
neg	0.87	0.86	0.86	6280
pos	0.86	0.86	0.86	6189
accuracy			0.86	12469
macro avg	0.86	0.86	0.86	12469
weighted avg	0.86	0.86	0.86	12469

Accuracy score = 0.8606143235223354

**vi. For C = 100**

Confusion matrix: [[5271 1009]

[1032 5157]]

	precision	recall	f1-score	support
neg	0.84	0.84	0.84	6280
pos	0.84	0.83	0.83	6189
accuracy			0.84	12469
macro avg	0.84	0.84	0.84	12469
weighted avg	0.84	0.84	0.84	12469

Accuracy score = 0.8363140588659876

**vii. For C = 1000**

[[5192 1088]

[1149 5040]]

	precision	recall	f1-score	support
neg	0.82	0.83	0.82	6280
pos	0.82	0.81	0.82	6189
accuracy			0.82	12469
macro avg	0.82	0.82	0.82	12469
weighted avg	0.82	0.82	0.82	12469

Accuracy score = 0.8205950757879541

As we have seen in the different values of C the accuracy score first increase and then started decreasing because of the two parameters in loss function and both have to be taken care, Hence we have to choose the C accordingly. In this case,  $C = 1$  is giving the best results.

## 5. Gaussian based Bayes classifier (Gaussian Discriminant Analysis)

We applied the same data preprocessing techniques as above.

This classifier takes the mean and variance as variable and calculating it will define a gaussian region for each feature column and combining all will give the predicted values for the test set.

Assuming two classes: neg and pos

$$p1(x/neg) = N(\mu_N, \sigma_N^2)$$

$$p1(x/pos) = N(\mu_P, \sigma_P^2)$$

$$p1(pos/x) = (p(pos)*p1(x/pos))/(p(pos)*p1(x/pos) + p(neg)*p1(x/neg))$$

$$p1(neg/x) = (p(neg)*p1(x/neg))/(p(pos)*p1(x/pos) + p1(neg)*p1(x/neg))$$

and the final result will be  $P = p1 * p2 * \dots * pn$  (for each feature)

Confusion matrix :

		[[5179 1101]			
		[1292 4897]]			
		Precision	recall	f1-score	support
	neg	0.80	0.82	0.81	6280
	pos	0.82	0.79	0.80	6189
	accuracy			0.81	12469
	macro avg	0.81	0.81	0.81	12469
	weighted avg	0.81	0.81	0.81	12469

Accuracy score = 0.8080840484401315

### 1.3 Hate Speech and Offensive Language Detection

We are using the given dataset “Thomas Davidsons’s dataset”.

After importing the CSV file we found out the column is divided as count, hate\_speech, offensive\_language, neither, class, and tweets.

As classes (0 - hate\_speech, 1- offensive\_language, 2- neither) are assigned and all the tweets are analyzed and assigned to each class by CrowdFlower but for machine learning, we have to preprocess the data for more clarity and less complexity.

```
vectorizer = TfidfVectorizer(tokenizer=tokenize,preprocessor=preprocess,ngram_range=(1, 3),
stop_words=stopwords, use_idf=True, smooth_idf=False, norm=None, decode_error='replace',
max_features=10000, min_df=5, max_df=0.75)
```

In preprocessing, initially, we remove three main content initially

- Long text URL
- Mentioned comments
- Unorganized space patterns
- Hashtags

Tokenize - Then the data is converted all in lowercase and stemmer(stemmer = PorterStemmer()) is applied.

Stemmer basically converts different words, which are derived from the same word, to a single word so that feature extraction will be easy.

(playing, play, plays ----- converted to “play”)

Stopwords - words that are not useful in feature extraction can be removed.

```
{‘ourselves’, ‘hers’, ‘between’, ‘yourself’, ‘but’, ‘again’, ‘there’, ‘about’, ‘once’, ‘during’, ‘out’,
‘very’, ‘having’, ‘with’, ‘they’, ‘own’, ‘an’, ‘be’, ‘some’, ‘for’, ‘do’, ‘its’, ‘yours’, ‘such’, ‘into’, ‘of’,
‘most’, ‘itself’, ‘other’, ‘off’, ‘is’, ‘s’, ‘am’, ‘or’, ‘who’, ‘as’, ‘from’, ‘him’, ‘each’, ‘the’, ‘themselves’,
‘until’, ‘below’, ‘are’, ‘we’, ‘these’, ‘your’, ‘his’, ‘through’, ‘don’, ‘nor’, ‘me’, ‘were’, ‘her’, ‘more’,
‘himself’, ‘this’, ‘down’, ‘should’, ‘our’, ‘their’, ‘while’, ‘above’, ‘both’, ‘up’, ‘to’, ‘ours’, ‘had’, ‘she’,
‘all’, ‘no’, ‘when’, ‘at’, ‘any’, ‘before’, ‘them’, ‘same’, ‘and’, ‘been’, ‘have’, ‘in’, ‘will’, ‘on’, ‘does’,
‘yourselves’, ‘then’, ‘that’, ‘because’, ‘what’, ‘over’, ‘why’, ‘so’, ‘can’, ‘did’, ‘not’, ‘now’, ‘under’, ‘he’,
‘you’, ‘herself’, ‘has’, ‘just’, ‘where’, ‘too’, ‘only’, ‘myself’, ‘which’, ‘those’, ‘i’, ‘after’, ‘few’, ‘whom’,
‘t’, ‘being’, ‘if’, ‘theirs’, ‘my’, ‘against’, ‘a’, ‘by’, ‘doing’, ‘it’, ‘how’, ‘further’, ‘was’, ‘here’, ‘than’}
```

IDF(inverse document frequency) - gives the feature count as a matrix.

#### Running the model :

In the running part, we divided the data for 10% test and 90% train data and perform 5 fold CV so that no data will be missed for the classifier to train and we took an average of that data.

We used the logistic regression process for the classifier to train.



	precision	recall	f1-score	support
0	0.44	0.59	0.51	164
1	0.96	0.91	0.93	1905
2	0.83	0.94	0.88	410
avg / total	0.91	0.89	0.90	2479

