Problem 1 : Naive Bayes

(a) Basic Implementaion

- Very basic data processing done:
  - Converted $[A..Z]$ to lowercase $[a..z]$
  - All symbols and numbers removed. All text was kept.
  - For example, `@Someone` and `#Covid19` used as `someone` and `covid` respectively
- Number of $Tweets$ in given data (True values)

Sentiment	Training Data	Validation data
Neutral	7096	617
Positive	16602	1444
Negative	14166	1232
Total	37864	3293

(i) Accuracies:

Test Set	Accuracy (%)
Training	80.044
Validation	70.270

(ii) Word Cloud (Training data on left , Validation data on right):

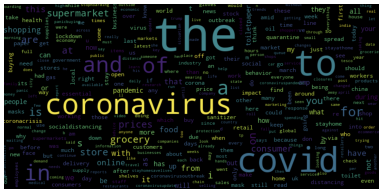
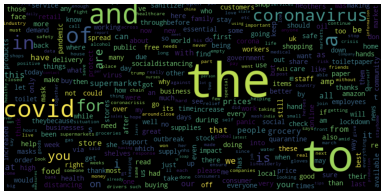
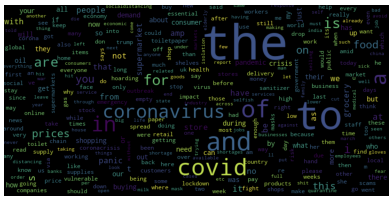
- Notice that there are a lot of `stopwords` like `a` `and` , `for` , `in` , `the` , `to` :
- Top words by frequency are shown below. Notice most of these don't carry any meaning/ give any sentiment information. These are stop words

Word	Frequency
`the`	41321
`to`	35442

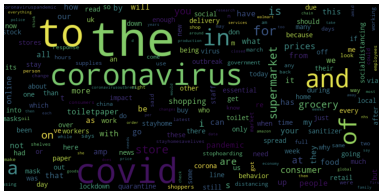
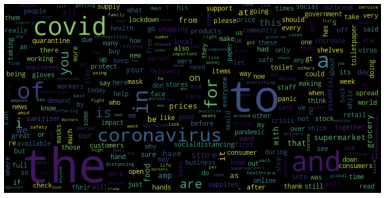
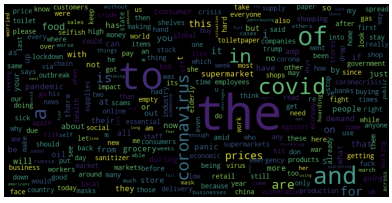
Word	Frequency
and	22171
covid	21381
of	19778
a	17988
in	17796
coronavirus	16770
for	12989
is	11295

Word Coluds:

Training Data

		
Neutral Tweets	Positive Tweets	Negative Tweets

Validation Data

		
Neutral Tweets	Positive Tweets	Negative Tweets

(b) Dumb Guessing

(i) and (ii) Accuracy (%) using various Algorithms:

Test Set	Naive Bayes	Random Guess	All positive
Training	80.044	33.604	43.846
Validation	70.270	33.951	43.851

(iii)

- On Validation set: `Naive Bayes` does 46.440% better than `Random Guess` and 36.198 % Better than `All positive`
- On Training set: `Naive Bayes` does 36.319% better than `Random Guess` and 26.419% Better than `All positive`

(c) Confusion Matrices:

- Training set size: 37864
- Validation set size: 3293
- Labels: Neutral(0) , Positive(1), Negative(2)
- Actual ↓ Predicted→

Training Set Validation Set

- `Random Guess` (note these will change each time the program is run)

	0	1	2		0	1	2
0	2339	2402	2355		203	203	211
1	5479	5566	5557		458	506	480
2	4724	4802	4640		405	413	414

- `All positive`

	0	1	2		0	1	2
0	0	7096	0		0	617	0
1	0	16602	0		0	1444	0
2	0	14166	0		0	1232	0

- `Naive Bayes`

	0	1	2		0	1	2
0	3463	2162	1471		172	278	167
1	296	14659	1647		27	1187	230
2	238	1742	12186		31	246	955

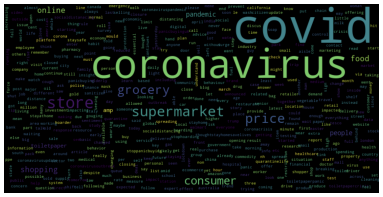
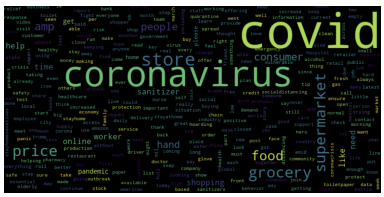
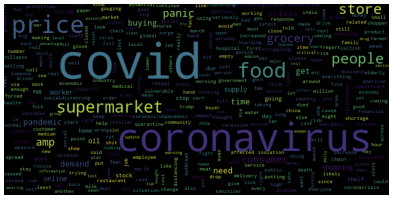
(ii) Highest value of diagonal entry: It's clearly in the `Naive Bayes` predictor, infact much higher than the `Random Guess` and `All positive` algorithms. Means `Naive Bayes` predicts more accurately and the independence assumptions made by it are reasonable to some extent

(d) Better Data Processing

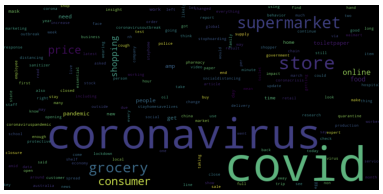
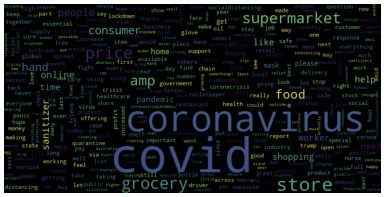
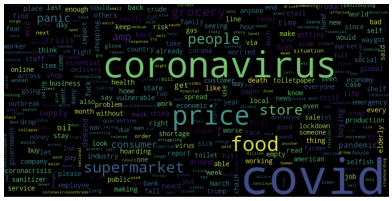
1. used `nltk :: stem :: WordNetLemmatizer()` to lemmatize worda. For example: - running , runs --> run
2. used `nltk :: corpus :: stopwords` to `remove stopwords`
3. Removed all possible `1 letter words` (`26` in number) and `2 letter words` (`676` in number, like `aa` , `cc` etc)
4. Removed all people references. For example `@HenrySmithUK` removed entirely as there didn't give any major sentiment information
   - Using this impreoved the accuracy from 70.179% to 70.999% on the validation set.
   - This reduced the training set accuracy from 80.303% to 79.741% which suggests that presence of `@people` was an overfitting feature

(ii) Word Clouds:

Training Data

		
Neutral Tweets	**Positive Tweets**	**Negative Tweets**

Validation Data

		
Neutral Tweets	Positive Tweets	Negative Tweets

(iii)

- Accuracy (%) after better data processing

Test Set	Accuracy(%)
Training	79.741
Validation	70.999

- It was seen that `Training Set` accuracy reduced on data cleaning, likely due to `overfitting` in the former case. Because words explained in `(3)` and `(4)` were quite specific to `few` or even `single` tweets only, so the model picked up the `noise` giving higher accuracy on `training set`
- (iv) Accuracy on validation set increased, as expected due to removal of noise and `pooling` words like { `run` , `runs` , `running` } that carry the same meaning together

(e) Feature Engineering

(i) Accuracy(%):

Test Set	Unigram	Unigram + Bigram	Unigram + Trigram	Unigram + Bigram + Trigram
Training	79.741	95.920	98.476	98.576
Validation	70.999	70.118	71.272	69.693

- `Bigram` feature clearly overfits. The training set accuracy shoots up and the validation set accuracy fals
- `Trigram` feature also looks like `slight` overfitting due to the high training accuracy. The validation set accuracy here is better then the previous
- Tried removing `#something` from text and considering them as a separate feature. This reduces validation accuracy.

- Used Puntuation marks ! ? separately as they convey Sentiment
- Accuracy (%)

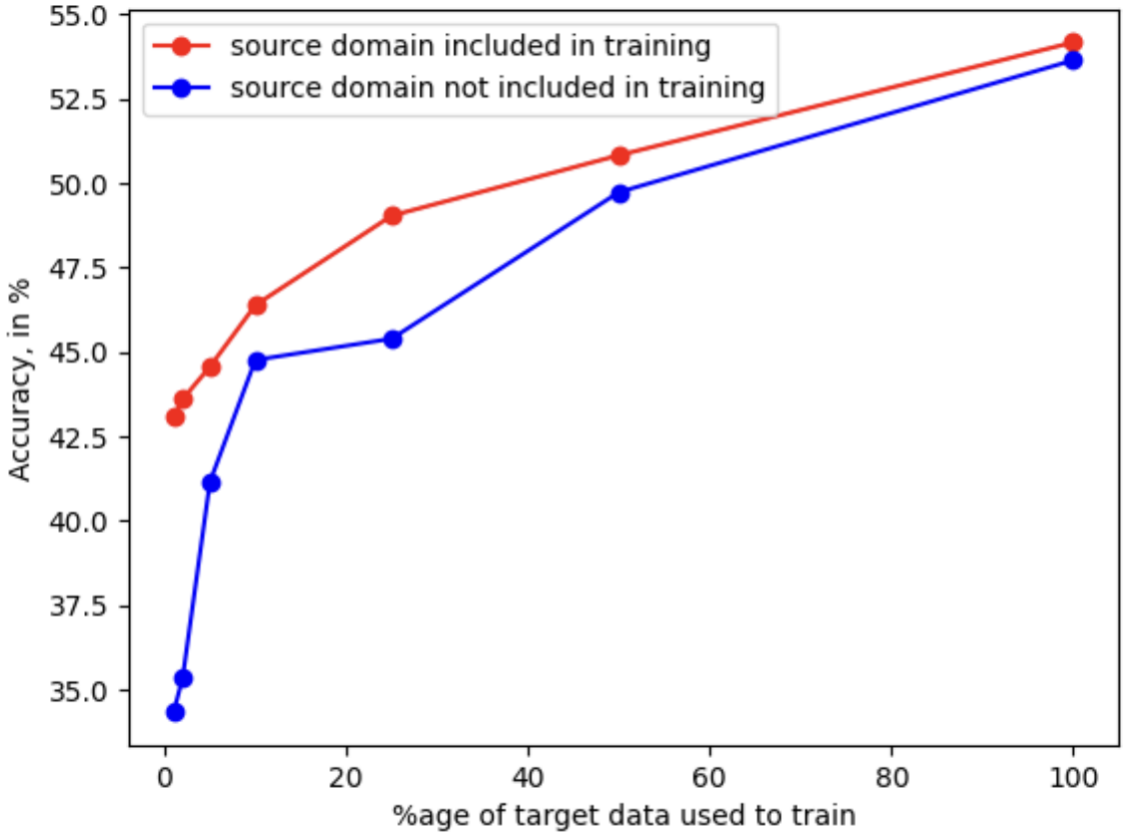
Test Set	[!?]	# totally removed
Training	79.725	78.977
Validation	70.938	70.817

Domain Adaptation

- Accuracy Values(%):

%age of Target Domain used to train	Including Sourch Domain for training	Excluding Source Domain in training
1	43.075	34.327
2	43.605	35.355
5	44.566	41.153
10	46.388	44.765
25	49.039	45.394
50	50.828	49.735
100	54.175	53.645

Plot:



- Adding Source data to training set out-performs Only target data in training set in all cases
- This suggests, even though the `general` Tweets in target domain are not completely related to the `CovidTweets` , adding Source Domain is better than learning the Target Domain from scratch. Further the data size if Target Domain is to small. Which leads to the relatively lower accuracies. The data of target domain only is a subset of source + target domain so no information is lost in the latter relative to the former

Problem 2 :: Binary Image Classification

Soft Margin SVM:

- The primal optimisaiton problem:

$$min_{\gamma,w,b}\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i$$

under:

$$y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, i = 1, 2...m$$

$$\xi_i \geq 0, i = 0, 1...m$$

- The lagrangian for the problem is:

$$L(w, b, \xi, \alpha, r) = \frac{1}{2}w^T w + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i[y^{(i)}(w^T x^{(i)} + b) - 1 + \xi_i] - \sum_{i=1}^{m} r_i\xi_i$$

with

$$r_i, \xi_i \geq 0, i = 1, 2...m$$

are the Lagrange multipliers

- The KKT conditions:

$$\frac{\partial}{\partial w}L(w, b, \xi, \alpha, r)|_{w^*, b^*, \xi^*, \alpha^*, r^*} = 0 \tag{1}$$

$$\frac{\partial}{\partial b}L(w, b, \xi, \alpha, r)|_{w^*, b^*, \xi^*, \alpha^*, r^*} = 0 \tag{2}$$

$$\frac{\partial}{\partial \xi}L(w, b, \xi, \alpha, r)|_{w^*, b^*, \xi^*, \alpha^*, r^*} = 0 \tag{3}$$

$$\alpha_i^*[1 - \xi_i^* - y^{(i)}(w^{*T} x^{(i)} + b^*)] = 0, i = 1, 2...m \tag{4}$$

$(1) \implies$

$$w = \sum_{i=1}^{m}\alpha_i^* y^{(i)} x^{(i)} \tag{9}$$

$(2) \implies$

$$\sum_{i=1}^{m}\alpha_i^* y^{(i)} = 0 \tag{10}$$

$(3) \implies$

$$\alpha_i^* + r_i^* = C, i = 1, 2...m \tag{11}$$

- Evaluating b^* Consider $i's$ such that:

$$0 < \alpha_i^* < C$$

- $\alpha_i^* < C$ and $(11) \implies r_i^* \neq 0$. Now $r_i^* \neq 0$ and $(7) \implies \xi_i^* = 0$
- $\alpha_i^* > 0$ $\xi_i^* = 0$ and $(4) \implies [1 - y^{(i)}(w^{*T}x^{(i)} + b^*)] = 0$
- noting that $y^{(i)} = \frac{1}{y^{(i)}}$ hence,

$$b^* = y^{(i)} - w^{*T}x^{(i)} \text{ whenever } 0 < \alpha_i < C$$

- The dual is:

$$max_\alpha \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j\langle x^{(i)}, x^{(j)}\rangle$$

  under

$$0 \leq \alpha_i \leq C$$

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

- Theoritically, b^{*} calculated from `any` support vector should turn out to be the same
- Here however, again due to limitation of `floating point arithmetic` , I took b^* to be average of the above quantity over all $x^{(k)}$ such that $\epsilon < \alpha_k < C - \epsilon$ for some suitable tolerance ϵ.
- Prediction is made using:

$$q = w^{*T}x + b^*$$

$$\hat{y} = 1 \ \ if \ \ q > 0 \ \ else \ \ -1$$

(a) Linear Kernel

(i) Because of constraints like `floating point accuracy` , we can't compare values exactly, we need to use tolerance.

- k^{th} training example is a `support vector` if and only if it satisfies:

$$\epsilon \leq \alpha_k \leq C - \epsilon$$

- Setting $\epsilon = 0.9$, calculating b^* over all valid k gives:

$$b^*_{min} = 0.6758676$$

$$b^*_{max} = 0.6758695$$

- Which are very close, in line with the above derivation. Finally, took average of all such b^*

$$b^* = 0.675868$$

- The following obtained:
  - Total training set size: 4760
  - SV only if $\alpha > \epsilon$

ϵ	nSV	$\frac{nSV}{m}$ %	Comment
10^{-9}	4760	100.00	Classifies all examples as suport vectors
10^{-8}	4760	100.00	-
10^{-7}	3858	81.05	-
10^{-6}	3001	63.05	nSV Almost non changing in this region
10^{-5}	2915	61.24	nSV Almost non changing in this region
10^{-4}	2897	60.86	nSV Almost non changing in this region

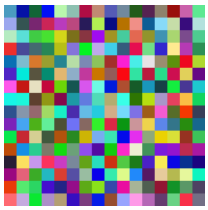
  - Choice of ϵ does not affect value of w^* and b^* later or any predictions. As explained before in claculation of b^*
  - $\epsilon = 10^{-6}$ looks reasonable. as the number of Support Vectors is almost the same for the last 3 values of ϵ (ii)
  - Validation set Accuracy: 71.750%

(iii)

- `Support vectors` corresponding to the top - 6 Coefficients:

$$\alpha = [0.99999998, 0.99999998, 0.99999999.\ 0.99999999.\ 0.99999999, 0.99999999]$$

- weight vector `w` :



(b) Gaussian Kernel

- b^* in this case can be calculated as:

$$b^* = -\frac{max_{i:y^{(i)}=-1} f(x^{(i)}) + min_{i:y^{(i)}=1} f(x^{(i)})}{2}, \alpha_i < C$$

where

$$f(x) = \sum_{i=1}^{m} exp(-\gamma ||x^{(i)} - x||^2)$$

- Prediction is made using:

$$q = b^* + \sum_{i=1}^{m} \alpha_i^* y^{(i)} \langle x^{(i)}, x \rangle$$

$$\hat{y} = 1 \ \ if \ \ q > 0 \ \ else \ \ -1$$

(i)

- Total training set size: 4760

ϵ	nSV	$\frac{nSV}{m}$ %	Comment
10^{-9}	4760	100.00	Classifies all examples as suport vectors
10^{-8}	4597	96.58	-
10^{-7}	3587	75.34	-
10^{-6}	3420	71.85	nSV Almost non changing in this region
10^{-5}	3406	71.55	nSV Almost non changing in this region
10^{-4}	3400	71.42	nSV Almost non changing in this region

- Number of SVs common in Linear and Gaussian Kernel: 2764, taking $\alpha = 10^{-6}$ (ii)
- Validation set accuracy: 77.250%
- This is better than the accuracy obtained by using Linear Kernel

(iii)

- `Support vectors` corresponding to the top - 6 Coefficients:

$$\alpha = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0]$$

- Notice 2^{nd} and 5^{th} are common with the SV's in linear Kernel

(c) SK Learn Classifier

- SK Learn:: Linear Kernel
- Scratch revers to the self built SVM Classifier
- taking $\alpha > 10^{-6}$ as the threshold to clasify as SV
- Notice: all `SV(SK Learn)` is a subset of `SV(scratch)`

Kernel	nSV Scratch	nSV SK Learn	nSV common
Linear Kernel	3001	2896	2896
Gaussian Kernal	3420	3397	3397
nSV common	2764	2684	

(ii)

- b^* learned

Model	Linear Kernel	Gaussian Kernel
from scratch	0.67586861	−2.72653712
sk learn	0.67620073	−2.72760044

- w^* for Gaussian Kernel is of infinte size so it can't be compared

- For Linear Kernel, the angle between $w^*_{scratch}$ and $w^*_{sklearn}$ is 0.09284° which is very close to 0. Angle caluclated using the relation

$$cos(\theta) = \frac{\langle \vec{v_1}, \vec{v_2} \rangle}{||\vec{v_1}|| * ||\vec{v_2}||}$$

(iii) Validation Accuracy (%)

- Surprisingly `scratch` achieved accuravy exactly same as `SK Learn's classifier` to 3 decimal places.

Model	Scratch	SK Learn
Linear Kernel	71.750	71.750
Gaussian Kernel	77.250	77.250

(iv)

- Training times: (seconds)

	Scratch	SK Learn
Linear Kernel	28.073	7.843
Gaussian Kernal	28.892	6.628

Problem3 :: MultiClass Classification

- Train data size: 14280 (all 6 classes combined)

(a) (b) Validation Set Accurary:

- SVM classifier built for every pair (total = 15) of classes.
- To make prediction, took votes from each pair. In case of tie used score
- Validation set Accuracy (%):

	Scratch	SK Learn
Accuracy	56.750	56.417

- Training time

	Scratch	SK Learn
Training time	8 minutes	1.5 minutes

(c) Confusion Matrices (on Validation Sets)

- Validation set size: 1200

Scratch

- Actual ↓ Predicted

Class	0	1	2	3	4	5
0	77	20	20	28	21	34
1	5	150	0	6	13	26
2	12	4	122	28	22	12
3	23	6	26	129	12	4
4	15	17	55	34	72	7
5	23	23$	7	8	8	131

SK Learn

- Actual ↓ Predicted

Class	0	1	2	3	4	5
0	78	20	20	27	22	33
1	5	150	0	6	14	25
2	12	4	122	28	22	12
3	24	6	25	129	12	4
4	17	17	57	34	68	7
5	24	23	8	8	7	130

- Class (4) images are most misclassified into class (2) images. The frequency is 55 in the Scratch Classifier and 57 in the SK Learn Classifier.

Class (4) images classified as class (2)

Correctly Classified Class (2) images:

- Most sparking observation is the large regions dominated by only one colour, blue or white which can be reason of the misclassfication. (4) is actually a set of water-related images. (4) is mountain related images. It's interesting to see in 7^{th} example in the former set, the "large background of water" likely causes the model to misclassify it into class (4) that is "large blue of sky" or "large white of snow mountains"

Mismatches in Scratch

					
4	4	5	0	2	0
3	1	0	4	5	5

- True Label in the upper row, Predicted Label in the lower
- class (1) has mostly trees and "green" which clearly suggests why the second sample is misclassified into (1)

Mismatches in SK Learn

					
0	5	1	4	4	4
4	0	4	5	3	5

- Some samples classified into (4) have a "large white/ blue background" like in (4)

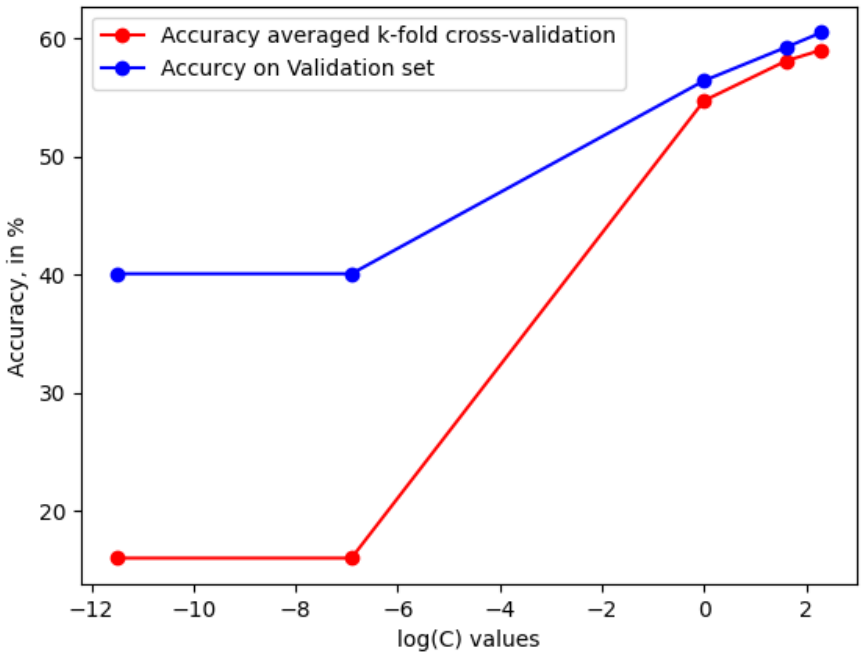
(d) K-fold Cross Validation

- Read the data. Merged the data of different classes into 1 array each of `X` and `Y`
- Shuffled the data for K-fold Cross validation, otherwise in one `block` there will a most samples of 1 class only which is wrong for training
- Validation Accuracies: Model trained on entire training data for different C values, then tested on Validation Data
- K Fold Accuracy: Training Data split into $k(=5)$ partitions, one partiiton excluded each time for testing, model trained on union of the remaining 4 partitions. Average accuracy over these k experiments reported.

C	K-Fold Cross Validation Accuracy (%)	Validation Ser Accuracies(%)
10^{-5}	15.74	40.08
10^{-3}	15.74	40.08
1	54.89	56.42
5	58.13	59.25
10	59.16	60.50

- Accuracy is maximum for $C = 10$ in both cases in infact monotinically
- Validation set accuracy (tested on Validation set) follows the same treand as K-fold accuracy (tested on fraction of Trainng set) as suggested in the theory.
- The best value of Accuracy in both is for the same C value
- In this way, we could have used a lesser amount of data for training in order to choose the most fitting value of C without haing to touch the Validation data. Overall, kesser data is used for training by K-fold cross validatio.

Plot:


Accuracy averaged k-fold cross-validation
Accurcy on Validation set
Accuracy, in %
60
50
40
30
20
−12
−10
−8
−6
−4
−2
0
2
log(C) values