CHIRAG MOHAPATRA
2018CS50403

---

1. In this problem, we had to implement the Naive Bayes algorithm as discussed in the class and use it for text classification on a subset of the Yelp dataset.

   (a) First I converted my dataset into appropriate training data. I removed the punctuations from each document and split them into a list of words. After that, I stored the words in a dictionary with each word being assigned a unique index and then each document just became a list of integers where $x_i = j$ means that word j is present in the i^{th} index of the document.

   After converting the dataset, I then calculated the parameters of the Naive Bayes model according to these formulas:

   $$\phi_k = \frac{\mathbb{1}\{y^{(i)} = k\}}{m} \tag{1}$$

   In 1, $y^{(i)}$ refers to the label of the i^{th} document and m refers to the total number of documents.

   Note: I have not explicitly stored ϕ_k, rather I have stored $log(\phi_k)$

   And,

   $$\theta_{l/k} = \frac{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} = k\} \sum_{j=1}^{n^{(i)}} \mathbb{1}\{{x_j}^{(i)} = l\} + 1}{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} = k\} n^{(i)} + |V|} \tag{2}$$

   In 2, $n^{(i)}$ refers to the number of words in the i^{th} document and $|V|$ refers to the total number of words in the vocabulary.

   Note: The 1 in numerator and V in denominator are added for Laplace smoothing

   Note: I have not explicitly stored $\theta_{l/k}$, rather I have stored $log(\theta_{l/k})$

   With this, the training of the Naive Bayes was complete. The time taken to train the raw model on my PC was around 110 seconds.

   Then, while making predictions for a test document x, I just had to do:

   $$prediction(x) = argmax_k(log(\phi_k) + \sum_{j=1}^{n} log(\theta_{l/k})) \tag{3}$$

   In 3, n refers to the number of words in the document.

   Note: These operations were all vectorized and optimized over numpy instead of a simple for loop

   For the model trained on just the raw words, I obtained:

   Train accuracy = 65.7292 %
   Test accuracy = 60.9574 %

   (b) Note that we had 5 different labels for the documents in our corpus, so just randomly guessing the label would yield:

   Test accuracy(random) = 20 %

   Also, in our corpus, the label 5 was the majority occurring class. Guessing each document to have label 5 would yield:

   Test accuracy(Max occurrence) = 43.9896 %

   Compared to these two, our test accuracy of 60.9574 % is considerably better.

(c) This is the test confusion matrix for the model trained on raw words(not stemmed):

14916 3038 1425 1086 2837
3018 2658 1284 455 192
1162 3120 4272 1662 345
639 1570 6521 18625 14408
434 452 1029 7530 41040

Note that for our confusion matrix here, the $[i][j]$ means that label i was predicted for true label j.

Label 5 is the one which has the highest value of the diagonal entry. This means that our model classified documents with label 5 correctly most number of times.

Note: Label 5 was also the most occurring label in both the train and test set

We can also see that the data with 1 and 5 stars have much greater accuracy of classification than $2, 3, 4$ which makes sense since reviews of 5 stars are all similar with multiple positive words and less scope of ambiguity.

(d) I performed stemming on the given documents to change the features and then trained a model on these new features.

Test Accuracy(Stemmed): 60.06072 %

The accuracy did not change very much and rather decreased over the test set. Mostly, stemming should help since we remove the stop words and also treat similar looking words as one. Probably this was some specific case in our test data because of which stemming did not help much.

(e) For feature extraction, I used a stemmed bigram model where I used bigrams of stemmed words as my features instead of just the words themselves. The test accuracy increased considerably in this case.

Test Accuracy(Stemmed Bigram): 64.09608 %

which is much better than our previous accuracy.

I also tested out a model with features the same as stemmed bigram except that it ignored words with length ≤ 3. For this model:

Test Accuracy(My features): 63.1508 %

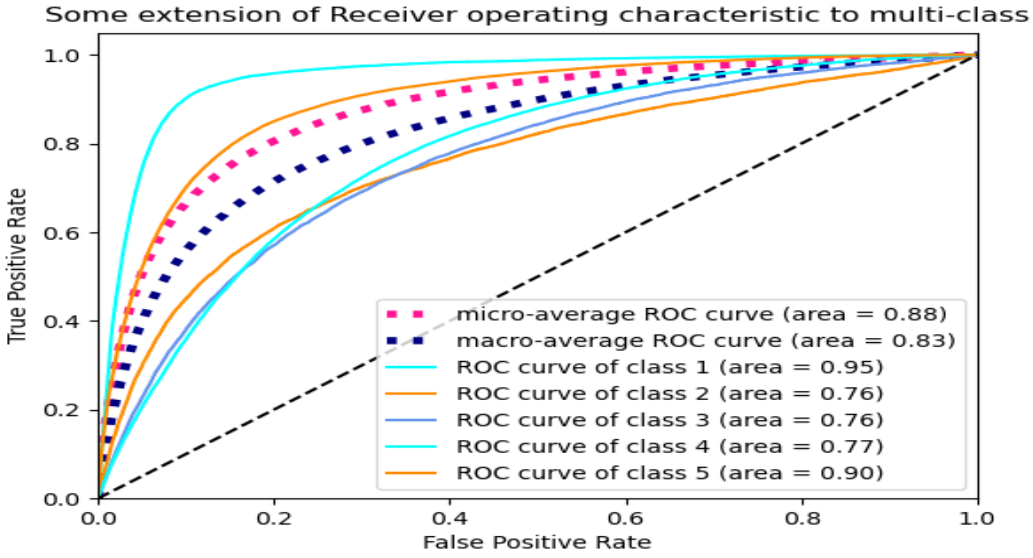
However, the number of features was reduced and the training of the model was completed faster.

Time to train the my features model: 72.22703719139099 seconds
Time to train the stemmed bigram model: 127.51838493347168 seconds

One thing that should be noted is that the main bottleneck is stemming and the stemming of train+test data takes almost 450 seconds.
The final model that I have submitted is the stemmed bigram model.

I also experimented with other features such as ignoring frequently occurring words(sort of tf-idf) but the results were not very promising.

(f) This is the ROC curve for the model trained on stemmed bigram features:



The area under curve is a maesure of how good the model is at classifying the data with 0.5 being no classification capacity and 1 being a perfect classifier.

We can see that the classification of 1 vs all and 5 vs all are much more accurate which also makes sense since review of 5 stars is supposed to have certain positive words and same with 1 and negative words.

Finally I have submitted the stemmed bigram model which is used for prediction of the test labels.

2. In this problem, we had to implement the Support Vector Machine algorithm as discussed in the class and use it for classification on a subset of the FMNIST dataset(for image classification).

   Note: I converted the pixel values from 0-255 to 0-1 by dividing all of them by 255

   (a) The last digit of my entry number is 3. So, I had to perform binary classification between labels 3(Dress) and 4(Coat).

   i As discussed in class for linear SVMs, the dual problem becomes:

$$\begin{aligned} max_\alpha \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)^T} x^{(j)} \\ subject\ to \quad & 0 \leq \alpha_i \leq c \quad \forall i \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{aligned} \tag{4}$$

   To solve this equation using CVXOPT, we have to convert this to the form:

$$\begin{aligned} min_x \quad & \frac{1}{2} x^T P x + q^T x \\ subject\ to \quad & Gx \leq h \\ & Ax = b \end{aligned} \tag{5}$$

   Our α is going to be a m x 1 matrix.
   With this in mind, I defined my matrices as:
   $P = yy^T * xx^T$

where $*$ denotes row wise multiplication while normal is the matrix multiplication.
This is a m x m matrix where: $P_{ij} = y^{(i)}y^{(j)}x^{(i)^T}x^{(j)}$
From 4, we can clearly see that this is the required P matrix.

q is a matrix of m x 1 with all elements -1.
Since we had the expression $-\sum_{i=1}^{m}\alpha_i$, so clearly q is our required matrix to get the desired expression.

G is a $2m$ x m matrix where we stack $-I$ on top of I where I is an identity matrix of m x m.
The $-I$ contributes to $\alpha_i \geq 0$ while the I contributes to $\alpha_i \leq c$.

h is a $2m$ x 1 matrix with the first m elements being all 0 and the last m elements being all c(Here, we have taken c to be 1.0).

A is just y^T and b is 0. This accounts for the equality $\sum_{i=1}^{m}\alpha_i y^{(i)} = 0$.

In this way, we have formulated the problem in a way acceptable by CVXOPT. Inputting to the solver, we get our dual objective α. Then, we filter out those α_i's which are almost 0(I have defined to be $\leq \epsilon = 1e-05$). The remaining α's are the dual objectives corresponding to support vectors. Filtering from the whole training set, we have obtained our set of support vectors!!

Then, using our dual, we calculate our primal objectives w and b as:
$w = \sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}$, here we only need the support vectors to calculate w.
We can calculate w as the sum of the rows of the matrix $\alpha * y * x$(this is a m x n matrix).
w will be a vector of n elements corresponding to the features.(Here, $n = 784$).
Then, we can calculate b as:

$$b = \frac{-1}{2}(min_{i:y^{(i)}=1}w^Tx^{(i)} + max_{i:y^{(i)}=-1}w^Tx^{(i)}) \tag{6}$$

With this, we have successfully trained our Linear SVM model!!
During prediction time, for given example x, if $w^Tx + b \geq 0$ then predict label 1 else predict label -1. With this model, I obtained:
Test Accuracy for linear svm: 94.5 %
Validation Accuracy for linear svm: 93.6 %
Note: These accuracies were obtained on the data with labels 3 and 4
We have obtained considerably high accuracies.

ii When we use a Gaussian Kernel, the equation becomes:

$$\begin{aligned} max_\alpha \quad & \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}\phi(x^{(i)})^T\phi(x^{(j)}) \\ subject\ to \quad & 0 \leq \alpha_i \leq c \quad \forall i \\ & \sum_{i=1}^{m}\alpha_i y^{(i)} = 0 \end{aligned} \tag{7}$$

Also, we have $\phi(x)^T\phi(z) = e^{-\gamma * ||x-z||^2}$ according to gaussian kernel, we have assumed $\gamma = 0.05$.
Looking at the equation, we can see that the only matrix for CVXOPT that we have to change from the linear case is P and all other matrices remain unchanged.
Now, we are aiming for a P such that $P_{ij} = y^{(i)}y^{(j)}e^{-\gamma * ||x^{(i)}-x^{(j)}||^2}$.
First I calculate the pairwise distance matrix D where $D_{ij} = ||x^{(i)} - x^{(j)}||^2$.
We can observe that:

$||x^{(i)} - x^{(j)}||^2 = (x^{(i)} - x^{(j)})^T(x^{(i)} - x^{(j)}) = {x^{(i)}}^T x^{(i)} + {x^{(j)}}^T x^{(j)} - 2{x^{(i)}}^T x^{(j)}$.
Consider g to be a m x 1 matrix where $g_i = {x^{(i)}}^T x^{(i)}$.
Then, we can represent D as:
$D = g\mathbb{1}^T + \mathbb{1}g^T - 2XX^T$ where $\mathbb{1}$ is a m x 1 matrix of all 1's.
Then, we can calculate our Gaussian matrix G with $G_{ij} = e^{-\gamma * ||x^{(i)} - x^{(j)}||^2} = e^{-\gamma * D_{ij}}$.
After this, we simply have $P = yy^T * G$.
Inputting into the solver, we get the dual objective α and then similarly as the linear case, we can get our support vectors by filtering out the α_i's.
For our primal objectives,
$w = \sum_{i=1}^{m} \alpha_i y^{(i)} \phi(x^{(i)})$. In this case, we don't explicitly calculate w.
For b, we have:

$$b = \frac{-1}{2}(min_{i:y^{(i)}=1} \sum_{j=1}^{m} \alpha_j y^{(j)} \phi(x^{(j)})^T \phi(x^{(i)}) + max_{i:y^{(i)}=-1} \sum_{j=1}^{m} \alpha_j y^{(j)} \phi(x^{(j)})^T \phi(x^{(i)})) \quad (8)$$

We can explicitly calculate b.
With this, we have successfully trained our gaussian kernel SVM model. At prediction time, for any test sample x, we will assign it label 1 if $\sum_{i=1}^{m} \alpha_i y^{(i)} \phi(x^{(i)})^T \phi(x) + b \geq 0$ and -1 otherwise.
We can do the prediction by calculating a similar pairwise distance matrix between our support vectors and the test samples to optimize the prediction function.

Test Accuracy for gaussian kernel svm: 96.1 %
Validation Accuracy for gaussian kernel svm: 96.0 %
Note: These accuracies were obtained on the data with labels 3 and 4
We have obtained better accurracies than the linear SVM case by using gaussian kernel.

(b) For this part, we work with the whole dataset for multi class classification.

i For this part, I trained 45 different gaussian kernel SVM classifiers.
The computational cost(training time) of this training is:
Training complete in 933.3501563072205 seconds on a high-end machine(*Note: The solving by CVXOPT ends up being the bottleneck here*).
While predicting for a sample x, it is predicted on all the 45 models and the label which gets the maximum number of predictions is the final predicted value of the sample.
The accurracies obtained are:
Test Accuracy(OnevsOne model): 85.05701140228045 %
Validation Accuracy(OnevsOne model): 85.03401360544217 %

ii Using the sklearn library, I trained a onevsone SVM classification model on the same dataset.
The computational cost(training time) of the algorithm is:
Training sklearn ovevsone complete in 198.23715567588806 seconds.
So, I observe that my model is approximately 4.7 times slower than the sklearn model.
Note: I discovered that the CVXOPT package very much depends on device capabilities because when training on a different low-end laptop, the onevsone training took almost 48 minutes while the sklearn still took only 300 seconds.
The accurracies obtained by the sklearn onevsone model are:
Test Accuracy(OnevsOne model) for sklearn: 88.07761552310463 %
Validation Accuracy(OnevsOne model) for sklearn: 87.91516606642658 %

iii The confusion matrices are as follows:
Test confusion matrix(my model):
[[404. 0. 0. 18. 0. 1. 56. 0. 1. 0.]
[0. 484. 0. 10. 1. 0. 1. 0. 0. 0.]

[7. 6. 414. 2. 56. 0. 57. 0. 1. 0.]
[7. 2. 4. 412. 14. 0. 6. 0. 0. 0.]
[0. 0. 26. 6. 365. 0. 20. 0. 0. 0.]
[0. 0. 0. 0. 0. 436. 0. 50. 1. 5.]
[71. 7. 43. 42. 52. 0. 345. 0. 3. 0.]
[0. 0. 0. 0. 0. 7. 0. 412. 0. 6.]
[10. 1. 13. 10. 12. 44. 15. 4. 494. 3.]
[0. 0. 0. 0. 0. 12. 0. 34. 0. 486.]]

Test confusion matrix for sklearn:
[[432 0 5 11 3 0 38 0 10 0]
[1 482 4 9 0 0 4 0 0 0]
[5 0 411 7 37 0 32 0 8 0]
[12 0 3 457 9 0 14 0 5 0]
[3 1 41 13 399 0 38 0 5 0]
[0 0 0 0 0 473 0 16 5 6]
[80 0 55 9 34 0 315 0 7 0]
[0 0 0 0 0 14 0 471 1 14]
[1 0 1 1 2 2 2 2 489 0]
[0 0 0 0 0 11 0 14 1 474]]

Validation confusion matrix(my model):
[[202. 0. 2. 12. 0. 0. 20. 0. 0. 0.]
[2. 240. 0. 10. 2. 0. 0. 0. 0. 0.]
[1. 2. 207. 0. 31. 0. 28. 0. 1. 0.]
[4. 2. 1. 195. 5. 1. 1. 0. 1. 0.]
[0. 0. 13. 6. 185. 0. 11. 0. 0. 0.]
[0. 0. 0. 0. 0. 228. 0. 27. 0. 2.]
[37. 3. 14. 23. 21. 0. 186. 0. 1. 0.]
[0. 0. 0. 0. 0. 1. 0. 198. 2. 4.]
[4. 3. 12. 4. 6. 14. 4. 4. 245. 5.]
[0. 0. 0. 0. 0. 6. 0. 21. 0. 239.]]

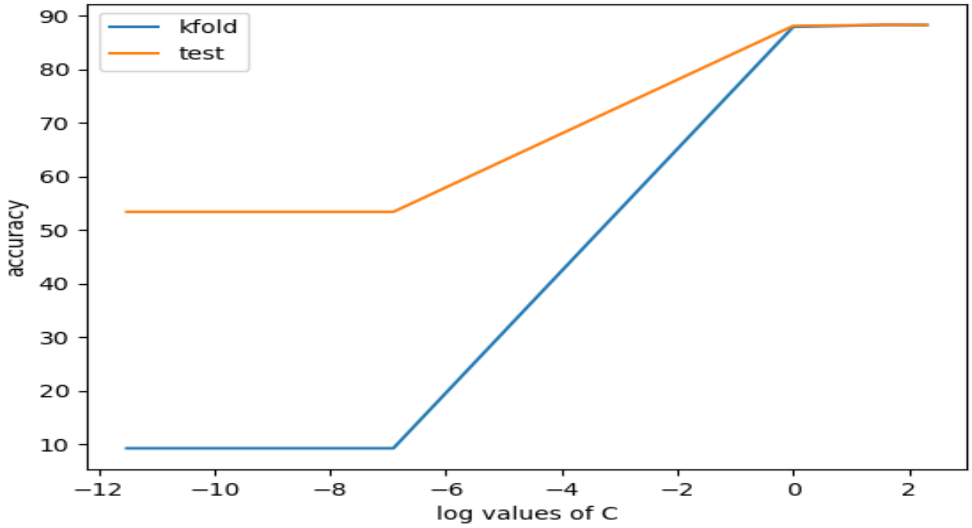
Validation confusion matrix for sklearn:
[[212 0 1 8 0 0 26 0 3 0]
[0 237 3 7 0 0 2 0 1 0]
[5 0 205 3 18 0 13 0 5 0]
[6 0 0 228 6 0 9 0 1 0]
[1 1 24 8 200 0 15 0 1 0]
[0 0 0 1 0 241 0 2 1 5]
[34 0 28 3 19 0 165 0 1 0]
[0 0 0 0 0 8 0 230 1 11]
[0 0 1 1 1 0 1 2 244 0]
[0 0 0 0 0 6 0 8 1 235]]

Looking at the confusion matrices, it is apparent that since our classifier provides high accuracy so obviously the diagonal elements are the greatest.
If we look at the non-diagonal elements, then in the test confusion matrix, we can see that $0, 6$ and $6, 0$ has some high values which makes intuitive sense since the classes in question are Shirts and T-shirts and there may be some ambiguity and mis-classification there.

iv I calculated the kfold cross validation accuracies on the given C values using scikit. These are the results:

Training kfold for C = 1e-05 complete in 3338.551198720932 seconds
k fold cross validation accuracy for C = 1e-05 : 9.293748240349705 %
Test Accuracy for C = 1e-05 : 53.39067813562713 %
Training kfold for C = 0.001 complete in 3344.0825679302216 seconds
k fold cross validation accuracy for C = 0.001 : 9.293748240349705 %
Test Accuracy for C = 0.001 : 53.39067813562713 %
Training kfold for C = 1 complete in 1010.8330547809601 seconds
k fold cross validation accuracy for C = 1 : 87.88835049764145 %
Test Accuracy for C = 1 : 88.07761552310463 %
Training kfold for C = 5 complete in 1105.4513666629791 seconds
k fold cross validation accuracy for C = 5 : 88.25726210762885 %
Test Accuracy for C = 5 : 88.27765553110622 %
Training kfold for C = 10 complete in 1109.2650980949402 seconds
k fold cross validation accuracy for C = 10 : 88.23947346323875 %
Test Accuracy for C = 10 : 88.2376475295059 %

This is the corresponding plot:



We can observe that $C = 5$ gives us the best kfold cross validation accuracy and this value also gives the best test accuracy(though it is very close between 5 and 10).
Also, we can observe that $C = 1e-05$ and 1e-03 are generally not acceptable values here since they give very low corresponding accuracies.
Using $C = 5$ with my own model gives test accuracy around 87.3% which is better than the one with $C = 1$.
So, we see how hyper parameter tuning is useful.

The final model that I have submitted uses a subset of training data as the validation set and then performs hyper parameter tuning to determine the parameter C from among $1, 5, 10$ using sklearn's onevsone model.(*Note: kfold is very slow and so I have not submitted it*).

Then, using the learned C and gamma=0.05, I then train my own onevsone model which is then used to predict the test labels.

3. For this problem, I first transformed my training data to the Count vectorised form where each document is composed of counts of the different words that occur in them. Then, I converted this count form to

a tf-idf vector for each document.(The tf-idf form ensures that common words like 'if','the' etc get less prioirity because of low idf and the tf ensures words which occur more number of times get higher values). With the training data in tf-idf form, I then applied the classification algorithms.

(a) NaiveBaeyes works better with just the count vector instead of tf-idf. Also, NaiveBaeyes is very fast.
Training of Naive Bayes complete in 0.6200966835021973 seconds
Test Accuracy of Naive Bayes model: 60.56327495176416 %

NaiveBaeyes is very fast, but the accuracy is not that good.

LinearSVC on the other hand did better with tf-idf form. I varied C values and these are some of the results:

C	Time Taken(in seconds)	Test Accuracy
0.1	26.91809	67.89587
0.5	53.4358	67.4022
1	85.4938	66.9506

LinearSVC provides the highest accuracy with a good value of C.

(b) I fixed max_iter to be 1000 and varied the tolerance first. These are the results of the SGD Classifier:

tol	Time Taken(in seconds)	Test Accuracy
1e-02	10.1495	63.1747
1e-03	11.1946	63.3033
1e-04	16.1998	63.3617
1e-05	36.8192	63.1799

After this, I fixed tolerance as $1e-04$(which gave highest accuracy) and then varied alpha. These are the results:

alpha	Time Taken(in seconds)	Test Accuracy
1e-02	10.4567	49.9013
1e-03	11.6975	57.6122
1e-04	16.5107	63.2832
1e-05	31.4987	65.6845

So, the SGD classifier works very well for a nice combination of tol and alpha. The loss is kept as hinge. Other losses did not perform so well.

My final conclusion is that the best model is the linear svc model where the parameter C is hyper-tuned using a validation set which is a subset of our training data. This is the model that I have submitted as the final model.