EECE5644 Machine Learning and Pattern Recognition
Homework 2

Rana Bogrekci

February 2024

You can access my codes from **here.**

Question 1

1

ML estimate of the mean:

$$\begin{aligned}\hat{\mu} &= \frac{1}{n} * \sum_{k=1}^{n} x_k \\ &= 1 * \sum_{k=1}^{1} x_1 \\ &= x_1\end{aligned}$$

2

Biased ML estimate of the variance:

$$\begin{aligned}\hat{\sigma}^2 &= \frac{1}{n} * \sum_{k=1}^{n} (x_k - \hat{\mu})^2 \\ &= 1 * \sum_{k=1}^{n} (x_k - x_1)^2 \\ &= (x_1 - x_1)^2 = 0\end{aligned}$$

3

Unbiased ML estimate of the variance:

$$\begin{aligned}\hat{\sigma}^2 &= \frac{1}{n-1} * \sum_{k=1}^{n} (x_k - \hat{\mu})^2 \\ &= \frac{1}{0} * 0 \ \textbf{undefined}\end{aligned}$$

4

Although we have a sample, the unbiased variance gives a result of undefined. The biased variance, however, is zero, as I expect. Normally, unbiased estimator would be preferable, but since it is undefined in this case, the biased estimator is more practical even though having one sample size results in zero variance.

Question 2

1

$$P(D|\theta) = p(x_1, x_2, ..., x_N|\theta)$$

Since samples in the dataset are i.i.d,

$$\begin{aligned}p(x_1, x_2, ..., x_N|\theta) &= p(x_1|\theta) * p(x_2|\theta) * p(x_3|\theta) * ..... * p(x_6|\theta) \\ &= \theta_{x_1} * \theta_{x_2} * \theta_{x_3} * ... * \theta_{x_6} \\ &= \prod_{k=1}^{K} p(x_i|\theta) \\ &= \prod_{k=1}^{K} \theta_k^{N_k}\end{aligned}$$

2

Considering the Dirichlet prior distribution,

$$p(\theta) \alpha \prod_{k=1}^{K} \theta_k^{\alpha_k - 1}$$

The posterior can be found by using Bayes' Theorem:

$$p(\theta|D) = \frac{p(D|\theta) * p(\theta)}{p(D)}$$

Since the p(D) here acts as a normalization constant and $p(D) = \int p(D|\theta)p(\theta)d\theta = 1$, the relation can be calculated from the numerator. Then,

$$p(\theta|D)\alpha \prod_{k=1}^{K} \theta_k^{N_k} * \prod_{k=1}^{K} \theta_k^{\alpha_k - 1}$$
$$\alpha \prod_{k=1}^{K} \theta_k^{N_k + \alpha_k - 1}$$
$$\alpha \prod_{k=1}^{K} \theta_k^{\alpha'_k - 1}$$

where

$$\alpha'_k = N_k + \alpha_k$$

3

If $\alpha_1 = \alpha_2 = ... = \alpha_K = 1$,

$$p(\theta)\ \alpha \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} = \prod_{k=1}^{K} \theta_k^{1-1}$$
$$= \prod_{k=1}^{K} \theta_k^0 = \prod_{k=1}^{K} 1$$
$$= 1$$

Then, $P(\theta)\alpha 1$ means every value of θ is equally likely. This is the uniform distribution, and the prior is uninformative.

4

It is known that $N = 1$ since we have one sample. Also, $\alpha_k = 1$ for all k = 1, ..., K.

For MLE,

$$\hat{\theta}_k^{ML} = \frac{N_k}{N} = N_k$$

For MAP,

$$\hat{\theta}_k^{MAP} = \frac{N_k + \alpha_k - 1}{N + \alpha_0 - K}$$

where

$$\alpha_0 = \sum_{k=1}^{K} \alpha_k = \sum_{k=1}^{K} 1 = K$$

Then,

$$\begin{aligned}\hat{\theta}_k^{MAP} &= \frac{N_k + \alpha_k - 1}{N + \alpha_0 - K} \\ &= \frac{N_k + 1 - 1}{1 + K - K} \\ &= N_k\end{aligned}$$

For the mean of the full Bayesian estimation,

$$\begin{aligned}E[\theta_k|D] &= \frac{N_k + \alpha_k}{N + \alpha_0} \\ &= \frac{N_k + 1}{1 + K}\end{aligned}$$

ML and MAP estimator gives the same result for uninformative prior, they reflect the observed sample without adding anything to it due to the uninformative prior. On the other hand, the result of the Bayes estimate points to the feature 'plus-one' smoothing which provides more information even though we have only one sample. Bayesian makes more sense because having one sample could lead to inadequate prediction of the posteriors, which MLE and MAP may do.

Question 3

1

a)

Naive Bayes classifier with a Dirichlet prior when parameters are learned via MAP estimation

$$g_c(x) = log\hat{\pi}_c + \sum_{j=1}^{d}\sum_{k=1} K_j 1_{x_j=k} * log\hat{\theta}_{jk|c}$$

where

$$\begin{aligned}\hat{\pi}_c &= \frac{N_c + \alpha_c - 1}{n + \sum_{c'} \alpha_c - K} \\ \hat{\theta}_{jk|c} &= \frac{N_{jk|c} + \alpha_{jk|c} - 1}{N_c + \sum_{k'} \alpha_{jk|c} - K_j}\end{aligned}$$

b)

In the case of full Bayesian, instead of point estimates, the posterior distributions are considered.

Naive Bayes classifier with a Dirichlet prior when parameters estimation is full Bayesian:

$$P(y = c|x)\alpha\bar{\pi}_c * \prod_{j=1}^{d}\prod_{k=1}^{K_j}(\bar{\theta}_{jk|c})^{1_{x_j=k}}$$

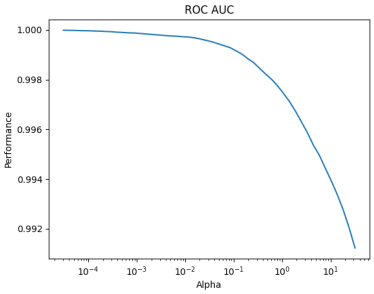
where

$$\bar{\pi}_c = \frac{N_c + \alpha_c}{n + \sum_{c'} \alpha_c}$$
$$\bar{\theta}_{jk|c} = \frac{N_{jk|c} + \alpha_{jk|c}}{N_c + \sum_{k'} \alpha_{jk'|c}}$$

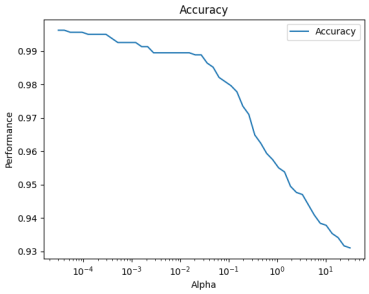
In the case where the prior is uninformative ($\alpha_c = 1$), MAP estimate becomes the same as ML estimate, on the other hand, full Bayesian estimate converges to a formula that has "plus-one" smoothing. With this smoothing, a count of one is added to each feature-class combination to avoid zero probabilities. This technique ensures that all features have non-zero probabilities even if they are absent in the training data, however it may be difficult to compute.

2

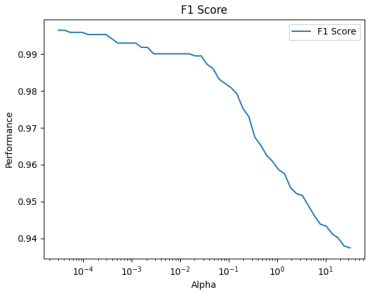
The mushroom dataset was processed with the $process - mushroom.py$ file. After that, a random split 80% of the dataset into training set and 20% into a test set was made. A categorical Naive Bayes classifier with different values of α was created. The range of α changes between 2^{-15} and 2^5. Plot of the predictive performance of the trained classifier by looking at the ROC AUC, accuracy, and F1 scores can be seen in Figure 1.



((a)) Graph of ROC AUC vs. Alpha



((b)) Graph of Accuracy vs. Alpha



((c)) Graph of F1 Score vs. Alpha

Figure 1: Predictive Performance vs. Alpha Graph

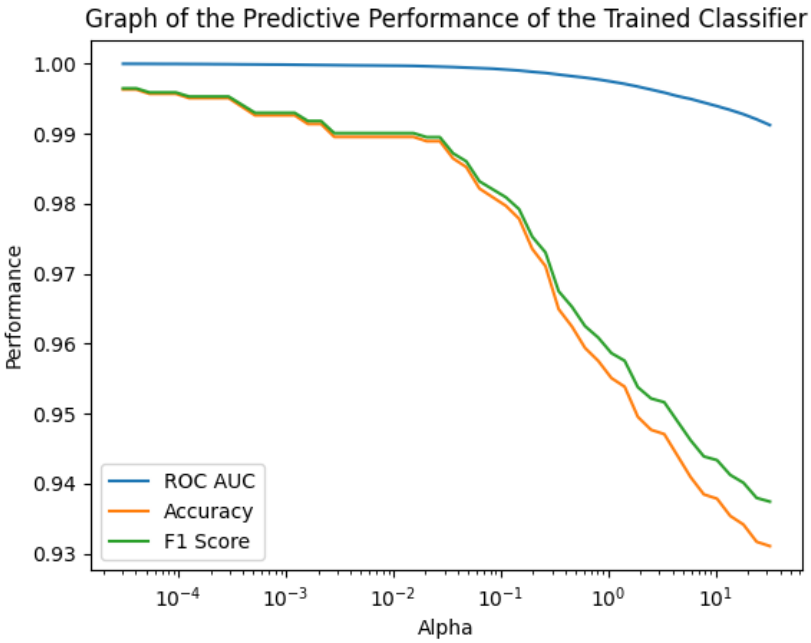


Figure 2: Graph of ROC AUC, Accuracy and F1 Score vs. Alpha

The α that maximizes the AUC is found as $3.0517578125 * 10^{-5}$.

The maximum ROC value reached with this alpha is 0.9999817772081467.

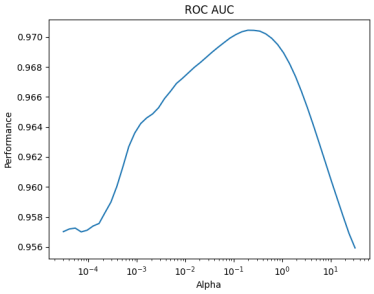
The negative log-likelihood parameters of the model in each label are as below.

[array([[-0.83255924, -4.46941331, -18.4501408 , -0.9268458 , -1.85242386, -6.95431063], [-0.78127017, -2.34676589, -5.07368321, -0.95736117, -2.87645993, -18.51541481]]),
array([[-1.02296014, -0.81085069, -1.63456809, -6.95431061], [-1.31359253, -1.02896391, -0.98411334, -18.51541479]]),
array([[-1.32910066, -1.77441165, -2.45822164, -1.58834478, -1.51624149, -3.77626659, -3.50963803, -18.45014084, -5.65503509, -18.45014084],
, -2.34676592, -1.78848618, -1.39317346, -1.89960696, -4.28956508, -4.40463433, -5.55325544, -4.86010943, -5.47914763
]),
array([[-1.83433288, -0.17402013], [-0.42017788, -1.06982063]]),
array([[-2.72021418, -18.45014083, -18.45014083, -3.47822183, -0.60111087, -2.9653366 , -1.94591017, -1.89383767, -4.58719627], [-18.51541484, -2.35301593, -2.3656344 , -0.20945238, -18.51541484, -18.51541484, -18.51541484, -18.51541484, -18.51541484]]),

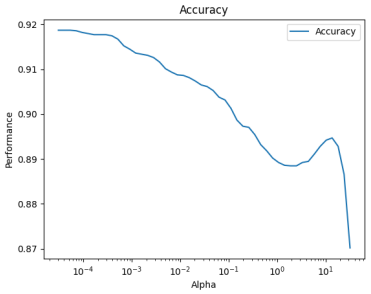
array([[-0.00574348, -5.1625596], [-0.04730097, -3.07478175]]),
array([[-0.03102027, -3.48858455], [-0.33082503, -1.26702211]]),
array([[-0.5573911 , -0.85027184], [-2.73830957, -0.06686612]]),
array([[-4.06394854, -3.48858464, -2.08167124, -1.79144141, -2.74962809, -2.03433986,
-4.41534619, -18.45014086, -0.81803471, -5.1625597 , -4.96188931, -18.45014086],
[-2.50507894, -1.46922257, -2.85551681, -1.61541707, -1.49281475, -3.00021316,
-2.23488469, -3.85552685, -18.51541487, -18.51541487, -4.22638624, -4.22638624]]),
array([[-0.72580974, -0.6615178], [-0.96047156, -0.48242019]]),
array([[-2.72021414, -4.36405287, -0.7531357 , -18.45014079, -0.79734179], [-
1.59758592, -2.10204986, -0.77931894, -3.08125429, -1.76732132]]), array([[-0.9316806
, -3.32554499, -0.56631975, -5.97348772], [-0.14333017, -2.33746348, -3.40867661,
-5.55325538]]),
array([[-0.94223693, -3.26544108, -3.81882613, -0.59763457], [-0.21387223, -2.22380417,
-3.00021309, -3.36461663]]),
array([[-0.81156682, -18.45014083, -1.12930447, -2.20649426, -2.20938862, -
18.45014083, -18.45014083, -4.58719627, -5.97348777], [-0.42746395, -1.98480902,
-1.98915685, -5.55325543, -18.51541484, -3.74875889, -3.07478182, -18.51541484,
-18.51541484]]),
array([[-0.84285347, -1.1059571 , -18.45014083, -2.20649426, -2.19499988, -
18.45014083, -4.96188928, -18.45014083, -4.58719627], [-0.44964601, -1.98915685,
-1.96973877, -18.51541484, -4.16696283, -3.7237576 , -18.51541484, -3.07478182,
-18.51541484]]),
array([[0.], [0.]]), array([[-2.54861283e-03, -1.84501408e+01, -1.84501408e+01, -
5.97348772e+00], [-4.73009875e-02, -3.74875885e+00, -3.78747334e+00, -1.85154148e+01]]),
array([[-0.02741667, -4.08264058, -4.58719621], [-0.13306314, -2.08272557, -
18.51541479]]), array([[-1.55465089, -0.79170797, -1.12243829, -18.45014079, -
4.58719623], [-0.28778146, -1.43234611, -18.5154148 , -4.48062013, -18.5154148
]]),
array([[-2.83257713, -2.8379972 , -18.45014083, -0.91565466, -0.7646887 , -
4.08264063, -18.45014083, -18.45014083, -18.45014083], [-0.92753107, -0.88475169,
-4.65247027, -4.48062017, -1.99352367, -18.51541484, -4.38053679, -4.48062017,
-4.53468735]]),
array([[-2.33920019, -18.4501408 , -18.4501408 , -0.31662599, -1.83234286, -
4.20278484], [-1.56000926, -2.35930523, -2.4044742 , -1.26702214, -1.36944752,
-2.6714696]]),
array([[-2.66386124, -1.67111704, -4.79483539, -1.13226157, -1.36707212, -18.45014081,
-1.85242387], [-3.86971144, -1.1051913 , -2.74292856, -0.80099468, -3.46424647,
-3.08776899, -2.8503488]])]

3

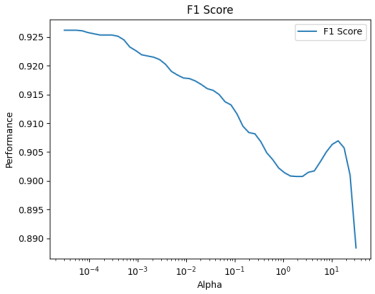
This time, a random split 1% of the dataset into training set and 99% into a test set was made. Plot of the predictive performance of the trained classifier by looking at the ROC AUC, accuracy, and F1 scores can be seen in Figure 3.



((a)) Graph of ROC AUC vs. Alpha



((b)) Graph of Accuracy vs. Alpha



((c)) Graph of F1 Score vs. Alpha

Figure 3: Predictive Performance vs. Alpha Graph

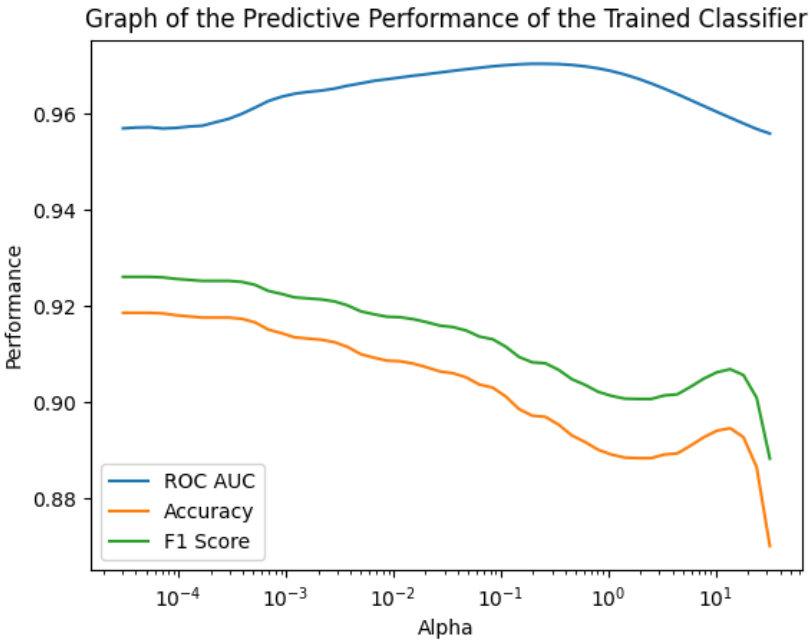


Figure 4: Graph of ROC AUC, Accuracy and F1 Score vs. Alpha

The α that maximizes the AUC is found as 0.19656266450237386.

The maximum ROC value reached with this alpha is 0.9704443581435691.

Inferring from these values, the maximum ROC AUC has decreased according to the training with more data. This can be because of the lack of the amount of the training data. Also, the alpha that maximizes the AUC has increased critically since it is trying to balance out the lack of training data compared to previous case.

From the figures, it can be said that the AUC starts with a very low value in the low training data case. However, as alpha increases, the effect of the training data diminishes; therefore, with the availability of the test data, the model learns to predict better until a point. After that point, alpha increases too much and oversmooths the decision boundary as it does in the previous case. Where in the previous case, the curve started from the top of the graph since there was enough training data and get high ROC for low alpha values. Nevertheless, the ROC was exponentially decreasing for the higher alpha values and the result of oversmoothing.

The negative log-likelihood parameters of the model in each label are as below.

[array([[-0.9850175 , -3.22117133, -5.02739832, -1.18179454, -1.29690945], [-0.5959166 , -2.80789152, -5.59674158, -1.12525807, -2.80789152]]),
array([[-1.56343119, -0.3819135 , -2.22534986], [-1.54691323, -1.11781054, -0.77630947]]),
array([[-1.01090294, -1.99224399, -2.63961606, -2.63961606, -1.20767998, -5.05328376, -5.05328376, -5.05328376, -3.24705677], [-1.8809834 , -3.19780457, -1.40474189, -1.02996705, -2.01109478, -3.80524528, -3.80524528, -5.61147228, -3.80524528]]),
array([[-2.5938671 , -0.07767022], [-0.39782696, -1.11406587]]),
array([[-5.05328376, -5.05328376, -5.05328376, -3.24705677, -0.7735099 , -3.24705677, -1.60251504, -1.60251504, -3.24705677], [-5.61147228, -2.82262222, -2.3367009 , -0.19605126, -5.61147228, -5.61147228, -5.61147228, -5.61147228, -5.61147228]]),
array([[-0.04156074, -3.20130781], [-0.02310299, -3.77932239]]),
array([[-0.04156074, -3.20130781], [-0.34255903, -1.23770612]]),
array([[-0.28082337, -1.40715731], [-3.17188167, -0.04282884]]),
array([[-5.07226739, -5.07226739, -3.2660404 , -2.01122762, -3.2660404 , -2.65859968, -5.07226739, -5.07226739, -0.43998097, -5.07226739, -3.2660404 , -5.07226739], [-2.1716109 , -1.49445147, -3.81615263, -1.5799988 , -1.34260576, -5.62237963, -2.1716109 , -3.81615263, -5.62237963, -5.62237963, -5.62237963, -3.81615263]]),
array([[-1.40715731, -0.28082337], [-1.00404415, -0.45632905]]),
array([[-5.02739832, -3.22117133, -1.42702082, -5.02739832, -0.3467853], [-1.55436076, -1.99636408, -0.783757 , -3.79051458, -1.75113779]]),
array([[-0.66635666, -5.01419992, -0.73442605], [-0.22018129, -1.85880518, -3.17562634]]),
array([[-0.74104703, -2.60715319, -2.60715319, -0.97844007], [-0.20048983, -1.99264725, -3.78679775, -3.78679775]]),
array([[-0.84014456, -5.04687495, -0.91894679, -2.63320724, -3.24064795, -5.04687495, -5.04687495, -3.24064795], [-0.44891077, -2.0074324 , -1.87732102, -5.6078099 , -5.6078099 , -3.19414219, -3.8015829 , -5.6078099]]),
array([[-0.84655338, -1.1044591 , -5.05328376, -3.24705677, -2.2644337 , -5.05328376, -3.24705677, -5.05328376, -3.24705677], [-0.48225195, -3.19780457, -1.48354412, -5.61147228, -3.80524528, -2.82262222, -5.61147228, -3.80524528, -5.61147228]]),
array([[0.], [0.]]),
array([[-0.00670983, -5.00753481], [-0.02310299, -3.77932239]]),
array([[-0.04822584, -5.01419992, -3.20797292], [-0.17387304, -1.85880518, -5.58929405]]),
array([[-2.23854826, -0.3467853 , -1.96635855, -5.02739832, -3.22117133], [-0.35375123, -1.24889832, -5.59674158, -5.59674158, -5.59674158]]),
array([[-5.05328376, -3.24705677, -5.05328376, -1.60251504, -0.32657233, -5.05328376, -5.05328376, -5.05328376, -5.05328376], [-0.97918586, -0.88476084, -5.61147228, -5.61147228, -1.76586849, -5.61147228, -5.61147228, -5.61147228, -3.80524528]]),
array([[-5.03393277, -5.03393277, -5.03393277, -0.0679587 , -5.03393277, -3.22770578], [-1.55806383, -2.00006715, -2.81159459, -1.25260139, -1.55806383, -2.32567327]]),
array([[-3.2341978 , -2.25157474, -5.0404248 , -0.69258154, -1.76565342, -5.0404248 , -1.76565342], [-3.79790706, -1.02262882, -2.81528399, -0.83335595, -5.60413405, -2.54309428, -3.19046634]])]

4

Accuracy is suitable when the class distribution is balanced. ROC AUC is better when the class distribution is not balanced, when the cost of false positives and false negatives varies significantly. In the case where mushrooms are decided if edible or poisonous, ROC AUC is more suitable. To explain, a false negative might not lead to a serious situation so the cost of that would be low. However, a false positive means that we would classify a poisonous mushroom as edible and the cost of this would be deadly high.

Question 4

By using the *process − sentences.py* file, the data was processed. Repeating 10 times, a random split of 80% of the dataset into a training set and 20% into a test set was created. After training a multinomial Naive Bayes classifier, different values for α within the range of 2^{-15} to 2^5 was used. Then, only the accuracy of the test set was measured. After 10 repeats, the average accuracy and standard deviation of the accuracy were calculated. Below, Figure 5 shows the average accuracy and its standard deviation.

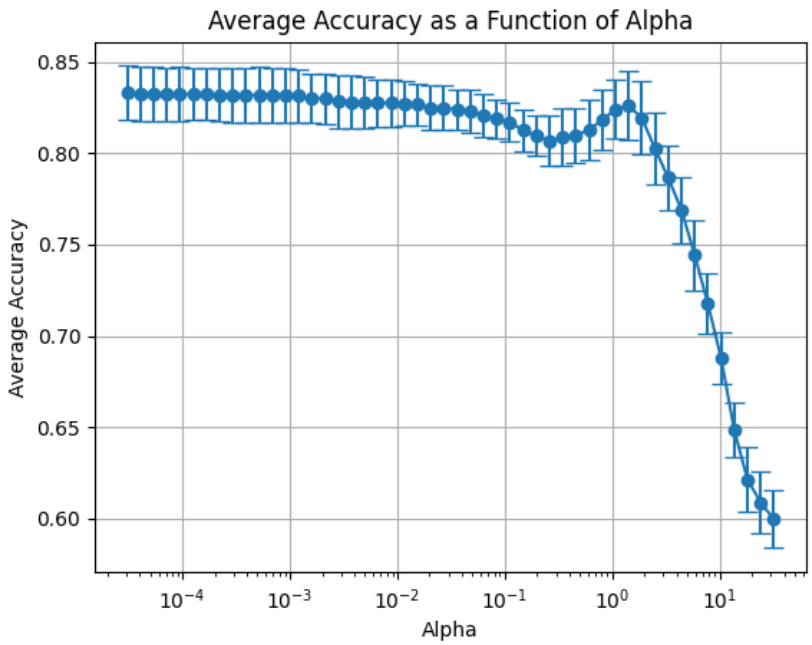


Figure 5: Graph of Average Accuracy and Confidence Intervals around the Average

The α that maximizes the average accuracy is found as $3.0517578125 * 10^{-5}$.

The maximum average accuracy reached with this alpha is 0.8330128205128207.

The words corresponding to the 5 highest parameters in each class are tabulated as below:

AIMX	we	to	a	of	the
OWNX	to	in	and	of	the
CONTRAST	citation	to	and	of	the
BASE	in	citation	and	of	the
MISC	and	citation	to	of	the

Table 1: 5 Highest Parameter for Each Class

The output of the *sentences.py* showing the results of alpha and words can be seen below.

```
  import pandas as pd
Best alpha value: 3.0517578125e-05
Maximum average accuracy: 0.8330128205128207
Top 5 words for the AIMX class:
we
to
a
of
the
Top 5 words for the OWNX class:
to
in
and
of
the
Top 5 words for the CONTRAST class:
citation
to
and
of
the
Top 5 words for the BASE class:
in
citation
and
of
the
Top 5 words for the MISC class:
and
citation
to
of
the
PS C:\Users\ranab\Desktop\PhD_2\eece5644\hw2>
```

Figure 6: Output of the Program - Best Alpha and Top Words

References

1. Mushroom. (1987). UCI Machine Learning Repository. https://doi.org/10.24432/C5959T.

2. Chambers, America. (2014). Sentence Classification. UCI Machine Learning Repository. https://doi.org/10.24432/C5D89S.

3. Stack Overflow

4. SciKit - Learn - Categorical Naive Bayes

5. SciKit - Learn - Multinomial Naive Bayes

6. Springboard - Data Analytics - Naive Bayes Classification

7. Analytics Vidhya - Blog - A Complete Guide to Dealing with Missing Values in Python