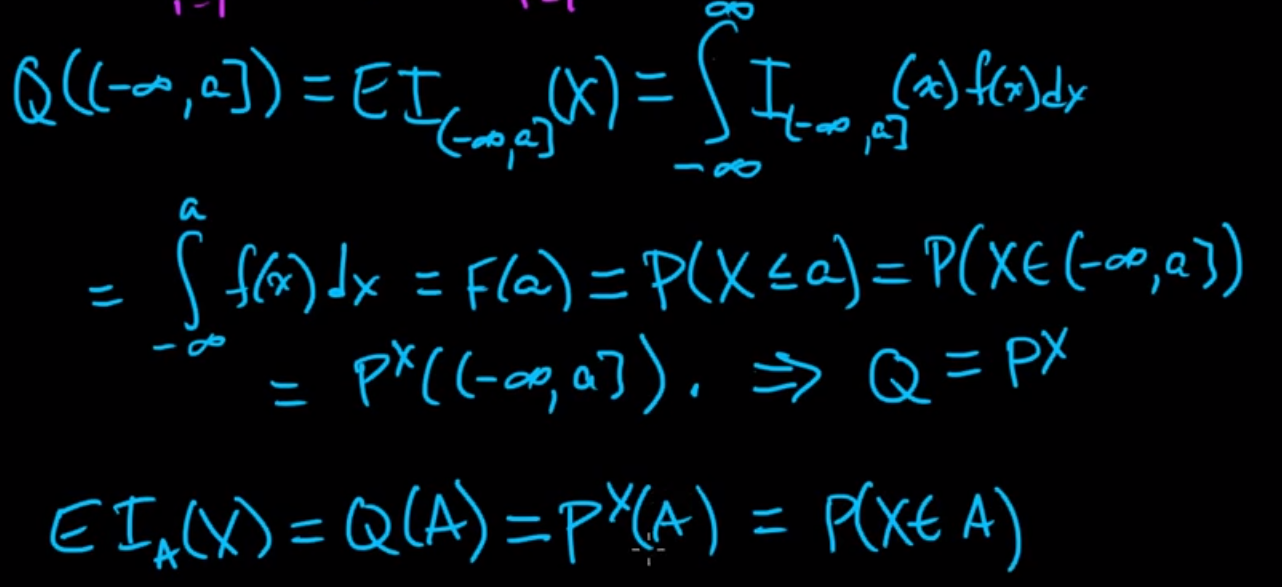
4a, b, c. Refer Notes

http://www.cs.cmu.edu/~10701/slides/10-701\_Fall\_2017\_Recitation\_2.pdf

5.

a. <https://www.youtube.com/watch?v=PiXu8_4X5dE>



b. https://www.youtube.com/watch?v=tFF2HXHNwcY

c.

the size grows exponentially with the dimension d.

2^784(number of bins^ dimensions)

<https://www.easycalculation.com/power-of-numbers.php>

d.

number of training points/region=k

e.

<https://people.eecs.berkeley.edu/~jfc/cs174/lecs/lec5/lec5.pdf>

<http://pages.cs.wisc.edu/~shuchi/courses/787-F09/scribe-notes/lec7.pdf>

6.a

Merge these 2 links below

<https://beginningwithml.wordpress.com/2018/09/18/7-gaussian-discriminant-analysis/>

<https://towardsdatascience.com/gaussian-discriminant-analysis-an-example-of-generative-learning-algorithms-2e336ba7aa5c>

Additional Ref <https://funglee.github.io/ml/slides/Lecture5-NaiveBayes-Notes.pdf>

6. b.Note that the two Gaussians have contours that are the same shape and orientation, since they share a covariance matrix Σ, but they have different means µ0 and µ1.

<https://svivek.com/teaching/lectures/slides/naive-bayes/naive-bayes-linear.pdf>

\textit{Based on question a and we substitute,}

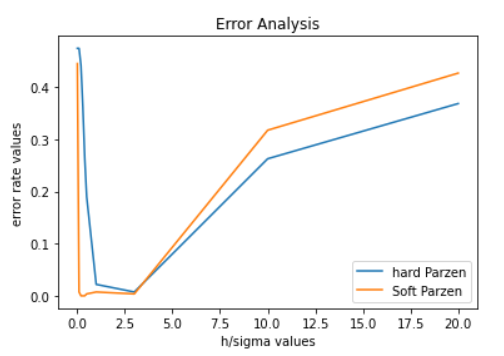
\[\theta\_1+\theta\_2+..+\theta\_N=1\]

\[(\frac{M\_1}{\mu\_N},\frac{M\_2}{\mu\_N}......\frac{M\_N}{\mu\_N})\*\theta\_N=1\]

\textit{We want $\theta\_N$}

\[\theta\_N=\frac{\mu\_N}{\sum\_{j=1}^{N}M\_j}\]

Q5. Give a detailed explanation on the below graph



I infer Bias Variance Trade-off in the error Analysis plot.

* In terms of hard Parzen, for h=0.01, there was high error rate as the model is not complex enough to capture the data points perfectly. As the h value slightly increase, the error rate drops drastically from 0.48 to around 0.05. As the h value increased further, there is a steep increase in the error rate after the h value greater than 3. So, from this graph the ideal rate to choose h would be around 3 with which we can get a better error rate with good model complexity so that our model learns rather than underfit or overfit.
* In terms of Soft Parzen, for sigma=0.01 similar to hard Parzen, the model underfits and the bias to much higher as the model couldn’t perfectly catch the points in the data. However, as the sigma value is slightly increased, the error is almost 0 for certain sigma values. At certain point, the error rates in soft Parzen window method sky rocketed for higher sigma values.

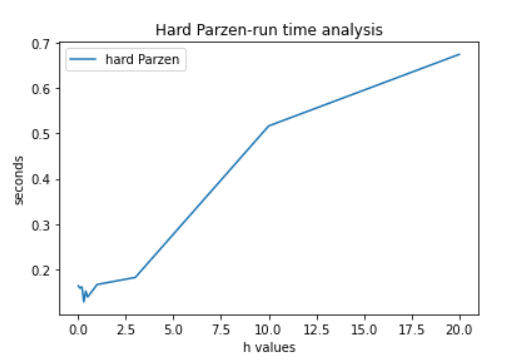
When we compare Hard and Soft Parzen error graph, the soft Parzen converged and provides slightly better error rate than Hard Parzen method. However, if the sigma values go up, if performs poorly compared to the hard Parzen. In my opinion, using Soft Parzen is preferred in terms of error rate. Let’s discuss further on the run time in Q7.

Q7.

Let us consider, m is the dimensions(features/columns) in the training set

n is the number of training examples.

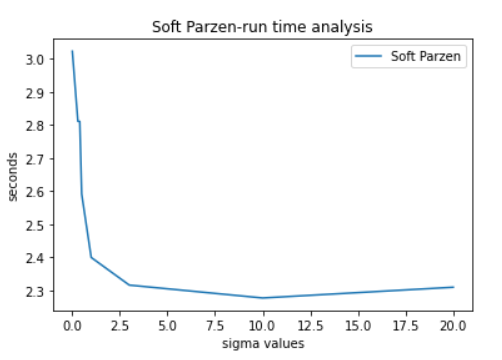
**Hard Parzen Runtime Complexity: O(m\*n)**



Hard Parzen suffers from the sparse-dimensionality curse i.e. as the dimensionality of the training set grows there would be larger volume in the hyperspace to capture the training points. The relationship is directly proportional, i.e. if the dimensionality grows the hyperspace grows. Hence, the distance between the points would increase and less points falls with the specified h i.e. less neighbours which might result in misclassification.

In terms of hyperparameter(h)🡪 As the h grows, the number of neighbours included with the region would be more, hence it results in some time to compute it. To summarize, as h grows the run time of the model grows.

**Soft Parzen Runtime Complexity: O(m\*n)**



Soft Parzen suffers from dense-dimensionality curse. For every iteration on the test point, it is computed with every point in the training set and finding the gaussian and multiplication makes soft Parzen computationally expensive (inner calculation of all the points) as it suffers from density. If the training points grows, the run time grows. When compared with Hard Parzen, the big O is same, however the based-on sparsity and density the run time varies between these two models.

In terms of hyper parameters (Sigma)🡪 As the sigma grows, there is no substantial increase in the run time as for every sigma value each test point is computed with every training point.

O(n)

<https://sebastianraschka.com/pdf/lecture-notes/stat479fs18/02_knn_notes.pdf>

https://towardsdatascience.com/k-nearest-neighbors-computational-complexity-502d2c440d5



This is prohibitively expensive for large number of samples But we need large number of samples for kNN to work well!

Soft🡪(n log n)

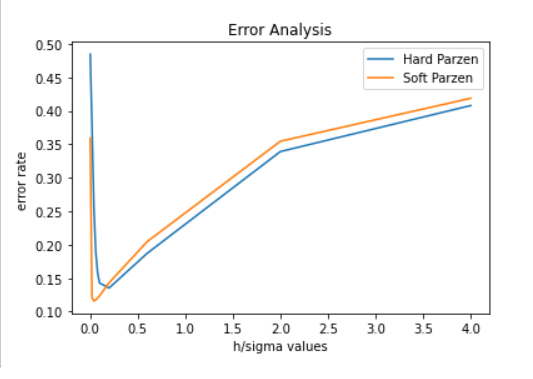
[0.48437956 0.38038686 0.25740146 0.18941606 0.15754745 0.14281752

0.13537226 0.18713139 0.33908029 0.40778832]

[0.35924818 0.12151825 0.1159854 0.11751825 0.12050365 0.12372993

0.14316788 0.20455474 0.35446715 0.41864964]

Q9.



When compared to the previous error rate analysis, its almost similar.

In terms of Bias Variance trade off🡪 When the hyper parameters values are less it couldn’t able to capture the data points well, as a result there was high error rates when h or sigma is equal to 0.01 and the optimal value for, hard Parzen(h)🡪 1 and soft Parzen(sigma)🡪 0.3. After this point, the error rate is high due to variance in the model i.e. overfitting.

In terms of run time🡪 By using random projections, we are reducing the dimension (from 4 to 2), it makes it better in terms of run time when compared to the previous result. However, in terms of Big O Notations it remains the same as O(m\*n).

To Summarize KNN algorithm,

* KNN also suffers Bias and variance problem, hence choosing the right hyperparameters would be essential for the model’s generalized learning.
* KNN mainly depends on the distance metrics (like Euclidean/Manhattan’s). Hence, I personally think, it might suffer from outliers and need as good amount of pre-processing.
* if we could reduce the dimensions through PCA or LDA or any other dimensionality approach, we could somewhat control the dimensionality curse in KNN algorithms.
* There is no as such training time for the KNN algorithms and the model doesn’t learn any parameters and mainly depends of the hyper parameters. Hence it is good to have a good amount of training points with dimensions reduced, so that model can perform well good.