ISYE 6740 HW 4

Mohammed Khalid Siddiqui

September 30, 2024

1 Conceptual Questions

1. Derive the gradient of the cost-function:

Homework 4

Given m training samples (x_i, y_i) $i = 1 \ldots m$,

$x_i \in \mathbb{R}$, $y_i \in \{0, 1\}$

we must solve the following to fit logistic regression model for classification

$$\max_{\theta} l(\theta)$$

where $l(\theta) = \sum_{i=1}^{m} \{-\log(1 + \exp\{-\theta^T x_i\}) + (y_i - 1)\theta^T x_i\}$

To find gradient, differentiate $l(\theta)$ w.r.t. θ

rule: $\frac{\partial l(\theta)}{\partial \theta} = \frac{1}{1 + \exp(-\theta^T x_i)} \cdot \exp(-\theta^T x_i) \cdot -x_i + (y_i - 1) x_i$

$$\nabla_\theta l(\theta) = \sum_{i=1}^{m} \left(\frac{x_i \exp(-\theta^T x_i)}{1 + \exp(-\theta^T x_i)} + x_i (y_i - 1) \right)$$

Sigmoid function: $g(z) = \frac{1}{1 + \exp(-z)}$

therefore $g(\theta^T x_i) = \frac{1}{1 + \exp(-\theta^T x_i)} = 1 - \frac{\exp(-\theta^T x_i)}{1 + \exp(-\theta^T x_i)}$

So that $\nabla_\theta l(\theta) = \sum_{i=1}^{m} ((1 - g(\theta^T x_i)) x_i + (y_i - 1) x_i)$

$$\sum_{i=1}^{m} ((1 - g(\theta^T x_i) + (y_i - 1)) x_i)$$

$1 - g(\theta^T x_i) = -(1 - g(\theta^T x_i))$

~~$1 - g(\theta^T x_i) = -(1 - g(\theta^T x_i))$~~

~~Therefore~~ $= \sum_{i=1}^{m} \{ (y_i - g(\theta^T x_i)) x_i$ ~~gradient = actual - predicted~~

$$= \sum_{i=1}^{m} \left(y_i - \frac{1}{1 + \exp(-\theta^T x_i)} \right) x_i$$

2. Pseudo-code to run gradient descent:

Pseudo-code for gradient descent to find θ^*

1. Set θ to some random value (or 0)
2. Set learning rate γ to some value
3. Set max iterations n (if needed)
4. Set a threshold for convergence G
5. For each iteration n :
6. initialize gradient = 0
7. For each training sample (X_i, y_i):
8. Compute predicted probability

$$\sigma(\theta^T x_i) = \frac{1}{1+\exp(-\theta^T x_i)}$$

9. Compute gradient for sample

$$\nabla_{\theta_i} l(\theta_i) \mathrel{+}= (\sigma(\theta^T x_i) - y_i) \cdot x_i$$

10. Update θ:

$$\theta_{n+1} = \theta_n - \gamma \nabla_\theta l(\theta)$$

11. Check for convergence:

if $|\theta_{n+1} - \theta_n| < \epsilon$

then break

12. Return θ

3. Pseudo-code to run stochastic gradient descent and the difference between SGD and GD:

3. Pseudo-code for Stochastic Gradient Descent

1. Initialize: θ, γ, number of epochs, G
2. For each epoch:
    3. Shuffle training data for randomness
    4. Divide data into S_n samples
    5. For each S_k in S_n:

6. For each training sample (x_i, y_i)

7. Compute $\sigma(\theta^T x_i)$

8. Compute $\nabla_\theta \ell(\theta) \pm (\sigma(\theta^T x_i) - y_i) x_i$

9. Update $\theta_{n+1} = \theta_n - \gamma \cdot \nabla_\theta \ell(\theta)$

11. Check for convergence $|\theta_{n+1} - \theta_n| < \epsilon$

if True, then break

The primary difference between GD and SGD is in SGD, we find the gradient for sample S to save on computing power, whereas in GD, we use the whole dataset. One of the drawbacks is that path to convergence may be noisier and less accurate for SGD as it uses a sample and sometimes even just one training example.

4. Using the Hessian to prove concavity for logistic regression training problem:

4. Concavity: $f(\theta x + (1-\theta)y) \geq \theta f(x) + (1-\theta) f(y)$

for all $0 \leq \theta \leq 1$

Given $l(\theta) = \sum_{i=1}^{m} (-\log(1+\exp(-\theta^T x_i)) + (y_i - 1)\theta^T x_i)$

- First-Order Derivative (gradient):

$$\nabla_\theta l(\theta) = \sum_{i=1}^{m} (y_i - \sigma(\theta^T x_i)) x_i$$

- Second-Order Derivative (Hessian)

↳ matrix of order-2 partial derivatives

$$H(\theta) = \nabla_\theta^2 l(\theta)$$

$$= \frac{\partial}{\partial \theta} \sum_{i=1}^{m} \left(y_i - \frac{1}{1+\exp(-\theta^T x_i)}\right) x_i$$

(Chain rule) $= \sum_{i=1}^{m} \left(\frac{1}{1+\exp(-\theta^T x_i)^2} \exp(-\theta^T x_i)\right) \cdot x_i$

$$= \sum_{i=1}^{m} \frac{x_i \cdot \exp(-\theta^T x_i) \cdot x}{(1+\exp(-\theta^T x_i))^2}$$

$$= \sum_{i=1}^{m} x_i \left(\frac{1}{1+\exp(-\theta^T x_i)} \cdot \left(\frac{1}{1+\exp(-\theta^T x_i)} - 1 \right) \right) \cdot x_i^T$$

$$= \sum_{i=1}^{m} x_i \cdot \left(\sigma(\theta^T x_i) \cdot (\sigma(\theta^T x_i) - 1) \right) \cdot x^T$$

$0 < \sigma(\theta^T x_i) < 1$

therefore

$$\sigma(\theta^T x_i) \cdot (\sigma(\theta^T x_i) - 1) \leq 0$$

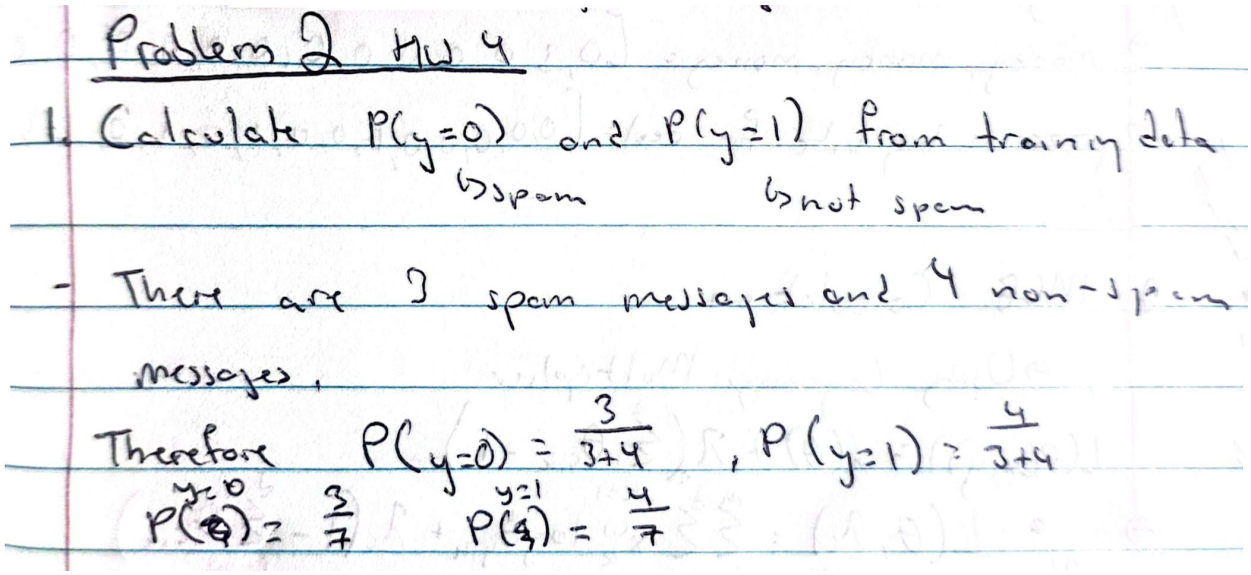
and of the form

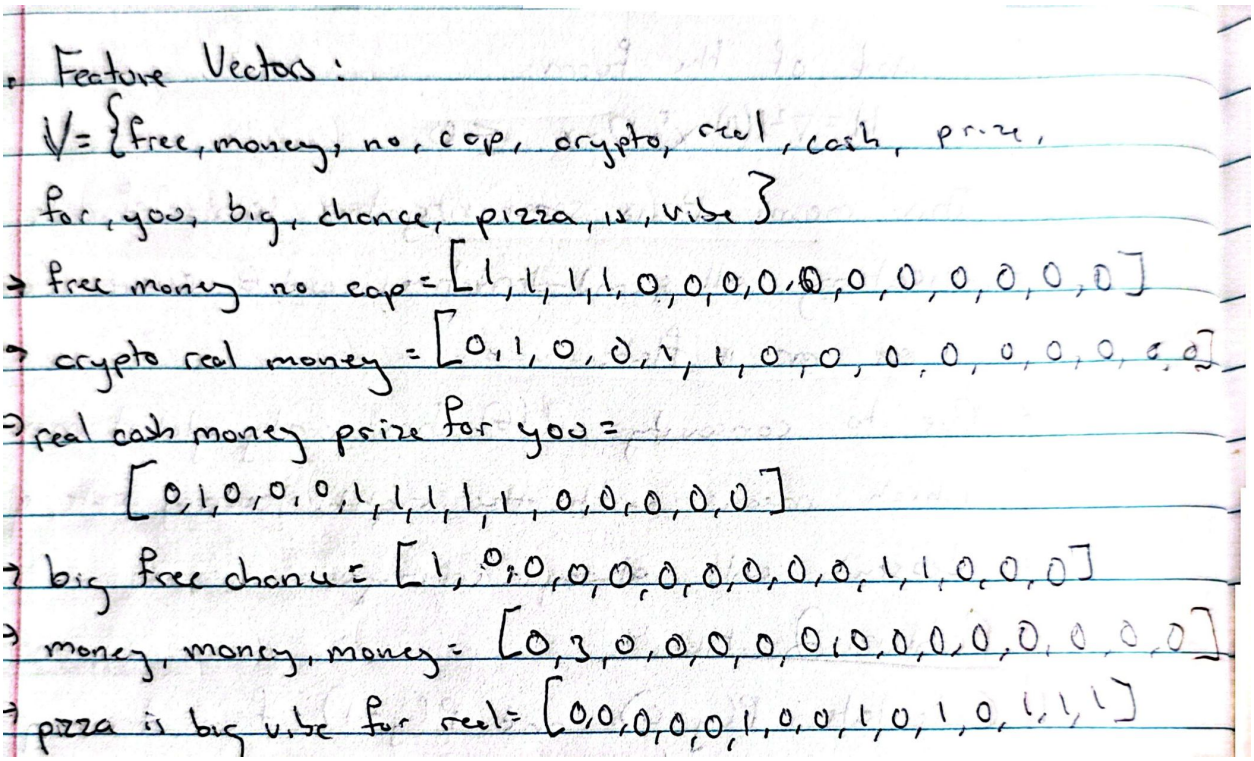
$$H = \nabla^2 l(\theta) = X^T \cdot D \cdot X \leq 0$$

- This means H is negative semi-definite indicating all second-derivatives wrt θ curves downward or is flat.
- Due to concavity, $l(\theta)$ has only global maximum which means with the right learning rate, gradient descent will always converge.

2 Bayes Classifier for spam filtering

1. Displaying all feature vectors and calculating priors:





2. Calculating MLE for Theta at four different combinations of class and word occurrences in respective class:

2. Likelihood of sentence ω its feature vector X given a class c is given by

$$P(x|y=c) = \frac{n!}{x_1! \cdots x_d!} \prod_{k=1}^{d} \theta_{c,k}^{x_k} \quad c = \{0,1\}$$

where $n = x_1 + \cdots x_d$ $0 \le \theta_{c,k} \le 1$ is probability of word k appearing in class c

Constraint $\sum_{k=1}^{d} \theta_{c,k} = 1 \quad c = \{0,1\}$

$$\ell(\theta_{0,1} \cdots \theta_{0,d}, \theta_{1,1} \cdots \theta_{1,d}) = \sum_{i=1}^{m} \sum_{i=1}^{d} x_k^{(i)} \log \theta_{y^{(i)},k}$$

Calculate MLE $\theta_{0,1}, \theta_{0,6}, \theta_{1,2}, \theta_{1,5}$

2. MLE Calculation

→ Using Lagrange Multiplier

$$L(P(x,y)) = \ell(\theta) + \lambda_c \left(\sum_{k=1}^{d} \theta_{c,k} - 1 \right)$$

→ • $L(\theta, \lambda_c) = \sum_{i=1}^{m} \sum_{k=1}^{d} x_k^{(i)} \log \theta_{y,k} + \lambda_c \left(1 - \sum_{i=1}^{d} \theta_{c,k}\right)$

$$\frac{\partial L}{\partial \theta_{c,k}} = \sum_{i=1}^{m} \frac{x_k^{(i)}}{\theta_{c,k}} - \lambda_c \quad \leftarrow \text{Set to } 0$$

$$\sum_{i=1}^{m} \frac{x_k^{(i)}}{\theta_{c,k}} - \lambda_c = 0$$

$$\theta_{c,k} \Sigma = \frac{1}{\lambda_c} \cdot \Sigma x_k^{(i)}$$

$$\sum_{k=1}^{d} \theta_{c,k} = 1 \rightarrow \sum_{k=1}^{d} \frac{1}{\lambda_c} \sum_{i=1}^{m} x_k^{(i)} = 1$$

$$\lambda_c = \sum_{k=1}^{d} \sum_{i=1}^{m} x_k^{(i)}$$

MLE:

$$\theta_{c,k} = \frac{\sum_{i=1}^{m} x_k^{(i)}}{\sum_{k=1}^{d} \sum_{i=1}^{m} x_k^{(i)}} \Bigg\} \text{ for given } c \in \{0,1\}$$

$c=1 \rightarrow$ non-spam $\quad c=0 \rightarrow$ spam

- $\theta_{0,1}$ ∈ Probability of word "free" appearing in spam messages
  - → free appears once in spam messages
  - → There are 13 word
  - MLE $\theta_{0,1} = \frac{1}{13}$
- $\theta_{0,6}$ ∈ Probability of "real" in spam messages
  - → "real" appears twice in spam messages.
  - → 13 word occurances in spam
  - MLE $\theta_{0,6} = \frac{2}{13}$
- $\theta_{1,2}$ ∈ "money" prob. in non-spam messages
  - → money appears 3 times
  - → 15 total word occurances in non-spam
  - MLE $\theta_{1,2} = \frac{3}{15} = \frac{1}{5} = 0.2$
- $\theta_{1,15}$ ∈ Prob. "vibe" in non-spam messages.
  - → "vibe" appears once in non-spam
  - → 15 total "non-spam" word occurances
  - MLE $\theta_{1,15} = \frac{1}{15}$

3. Determining the class using our Naive Bayes implementation for a given test message:

3. Test message: "money for real"

↳ Use NB to calculate posterior and classify as spam or non-spam

$x_{test} = [0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0]$

$P(y=0 \mid x_{test})$ ← spam

$P(y=1 \mid x_{test})$ ← non-spam

Bayes Theorem:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

$$P(y=0 \mid x_{test}) = \frac{P(x_{test} \mid y=0)\,P(y=0)}{P(x_{test})}$$

$$P(y=1 \mid x_{test}) = \frac{P(x_{test} \mid y=1)\,P(y=1)}{P(x_{test})}$$

$$P(x_{test}) = P(x_{test} \mid y=0)P(y=0) + P(x_{test} \mid y=1)P(y=1)$$

$P(y=0) = \frac{3}{7}$ $P(y=1) = \frac{4}{7}$

$P(x_{test} \mid y=0) = P(\text{"money"} \mid y=0) \cdot P(\text{"for"} \mid y=0) \cdot P(\text{"real"} \mid y=0)$

→ Use the MLEs

$\theta_{0,2} = \frac{3}{13}$ $\theta_{0,6} = \frac{2}{13}$ $\theta_{0,9} = \frac{1}{13}$

$P(x_{test} \mid y=0) = \frac{6}{2197}$

$P(x_{test} \mid y=1) = P(\text{"money"} \mid y=1) \cdot P(\text{"for"} \mid y=1) \cdot P(\text{"real"} \mid y=1)$

$\theta_{1,2} = \frac{1}{5}$ $\theta_{1,6} = \frac{1}{15}$ $\theta_{1,6} = \frac{1}{15}$

$P(x_{test} \mid y=1) = \frac{1}{3375}$; $P(x_{test}) =$

- We want to calculate whether $P(y=0 \mid x_{test})$ or $P(y=1 \mid x_{test})$ is larger.

- Since $P(x_{test})$ is the same denominator for

both, we can choose to ignore it as proportional relationship will be maintained, or we can calculate posterior.

$$P(y=0 \mid x_{test}) = \frac{\frac{6}{2197} \cdot \frac{3}{7}}{\left(\frac{6}{2197}\right)\left(\frac{3}{7}\right) + \left(\frac{1}{3375}\right)\left(\frac{4}{7}\right)}$$

$$= \frac{\frac{18}{15379}}{\frac{18}{15379} + \frac{4}{23625}}$$

$$\sim 0.8736$$

$$P(y=1 \mid x_{test}) = \frac{\frac{1}{3375} \cdot \frac{4}{7}}{\left(\frac{6}{2197}\right)\left(\frac{3}{7}\right) + \left(\frac{1}{3375}\right)\left(\frac{4}{7}\right)}$$

$$\sim 0.12637$$

Given that the posterior for $P(y=0 \mid x_{test})$ is greater than $P(y=1 \mid x_{test})$, we can conclude that our model predicts the message to be classified as spam.

3 Comparing classifiers: Divorce classification/prediction

1. For this problem, we predict whether a participant in a personal information form is divorced based on 54 predictors and 1 ground truth label where 1 means divorce and 0 represents not divorced.

   The three classifiers employed in this task are the standard sklearn implementations of Naive Bayes, Logistic Regression, and K-Nearest Neighbors.

   The success metric for our prediction is the testing accuracy, measured using the accuracy_score implementation within sklearn.metrics applied on the 20% test-set. For robustness, I applied Monte Carlo Cross Validation to split the dataset and run the algorithms 1000 times returning the mean test accuracy and variance for each classifier as well as the variance in the testing accuracy across the 1000 iterations. The data was also scaled prior to implementation as we are using KNN with a euclidean distance metric.

   Below, you'll see the distribution of test accuracy along with the mean accuracy and variance of the accuracy for each of the classifiers.

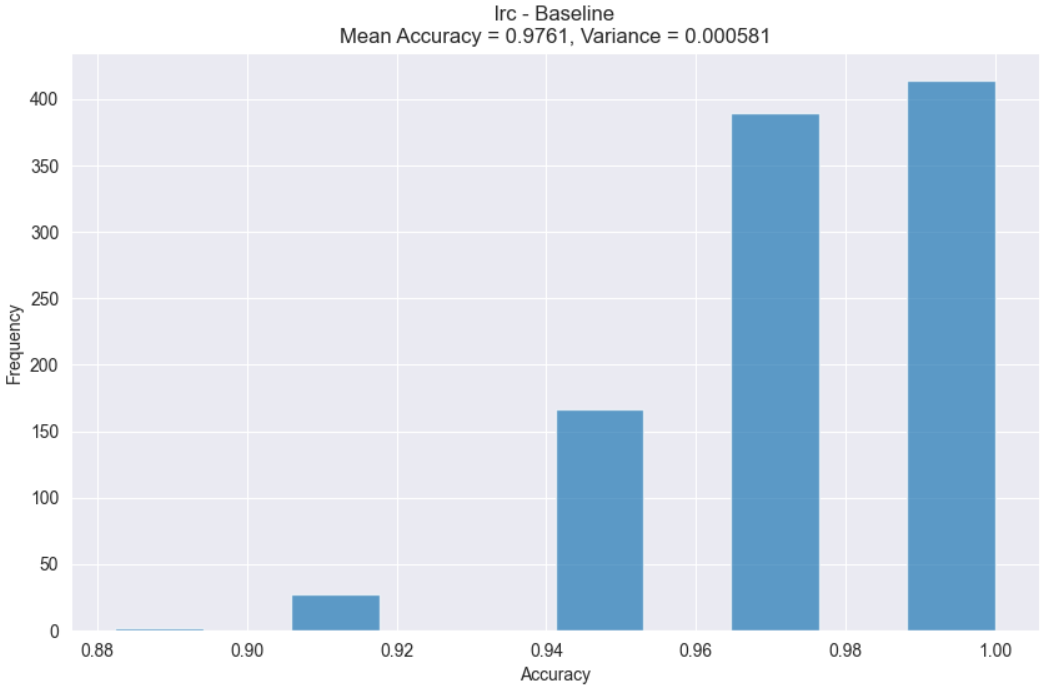


Figure 1: Logistic Regression

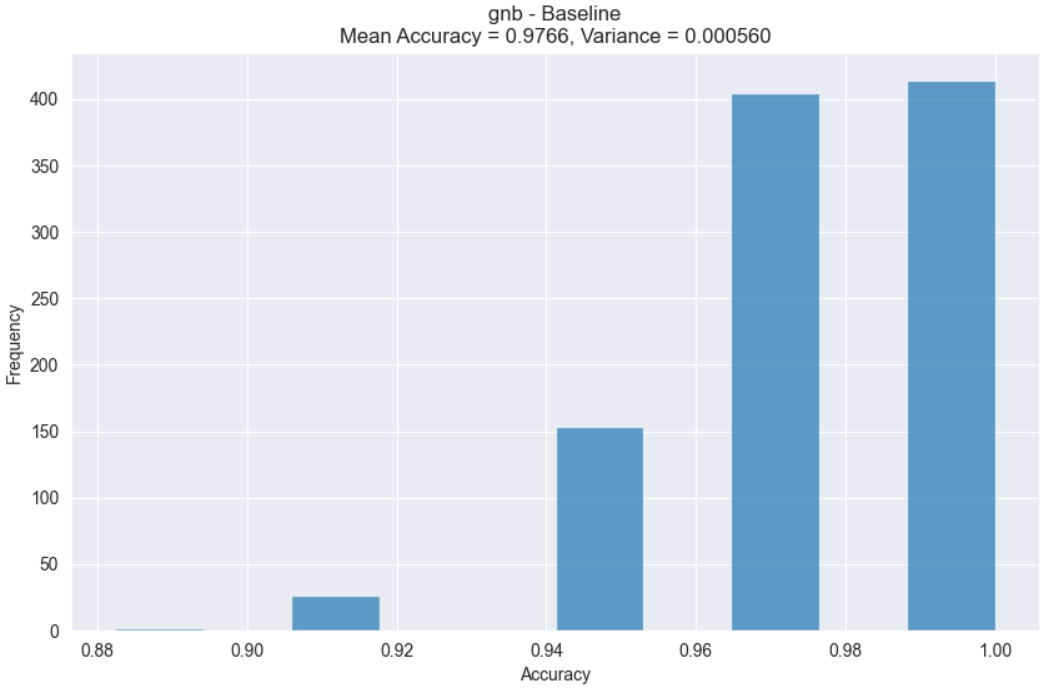


Figure 2: Naive Bayes

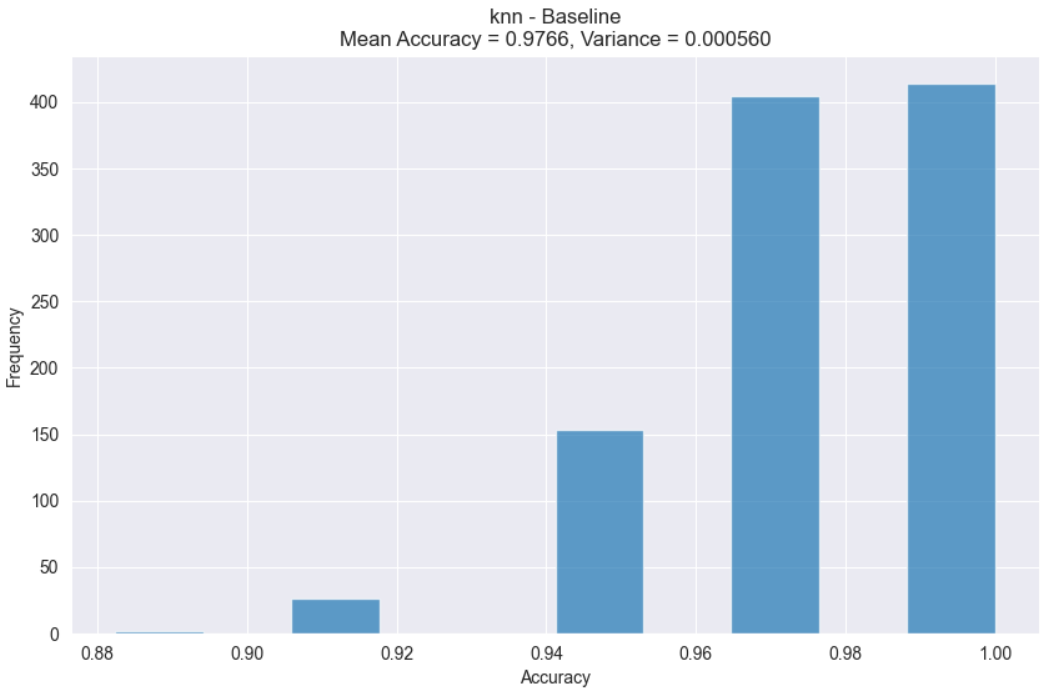


Figure 3: K-Nearest Neighbors

It seems the standard implementation of KNN (with five neighbors and using euclidean distance metric) and Naive Bayes (with a suggested smoothing parameter of 1e-3 to prevent divide by 0 errors arising where there is no variance among predictors) are tied for the mean accuracy and variance performing slightly better than our logistic regression model.

Metric	**Logistic Regression**	**Naive Bayes**	**KNN**
Mean Accuracy	0.9761	0.9766	0.9766
Variance	0.000581	0.000560	0.000560

KNN and Naive Bayes may perform better than Logistic Regression when decision boundaries are non-linear. Naive Bayes performs better when features can be assumed (almost) independent while KNN performs best when local relationships dominate and an appropriate distance metric can be used to map relationships between data points. Naive Bayes and KNN both perform well in low-dimensional, well-separated spaces.

2. The next step in this problem was to use PCA (I used the sklearn implementation with SVD) to reduce the dimension spaces to 2 before running the same Monte Carlo CV and returning the testing accuracy and variance for each.

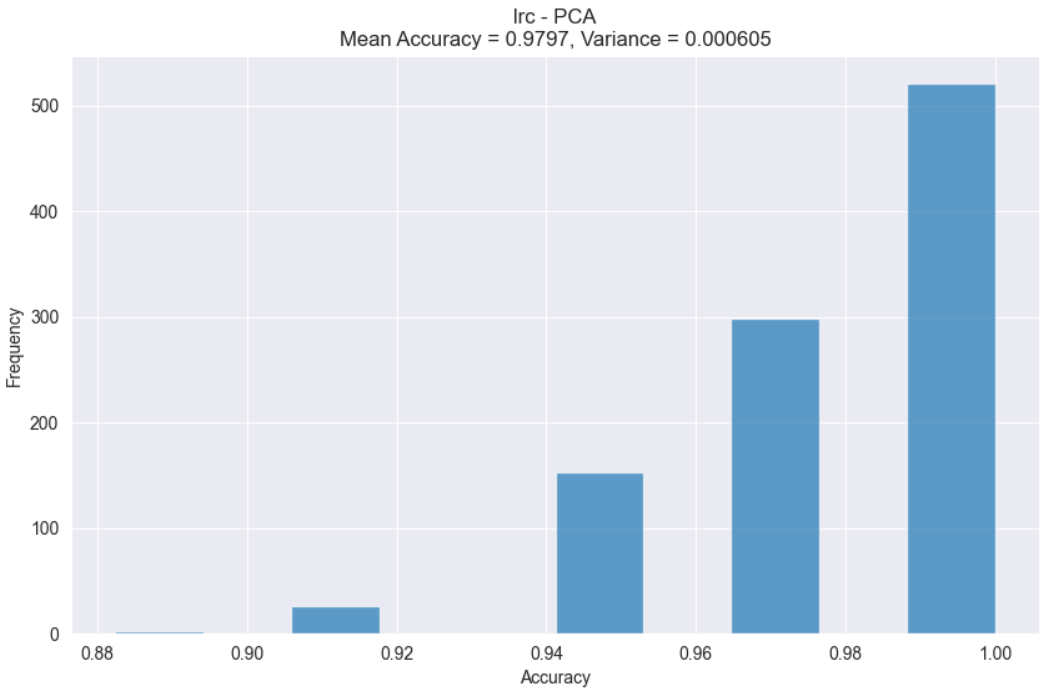


Figure 4: Logistic Regression

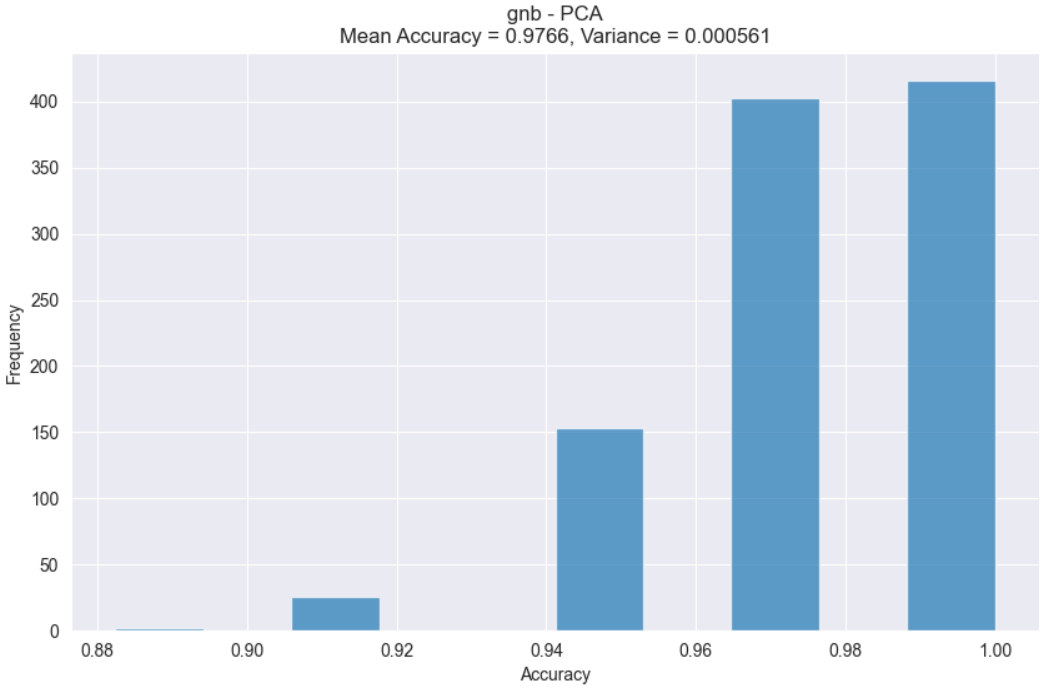


Figure 5: Naive Bayes

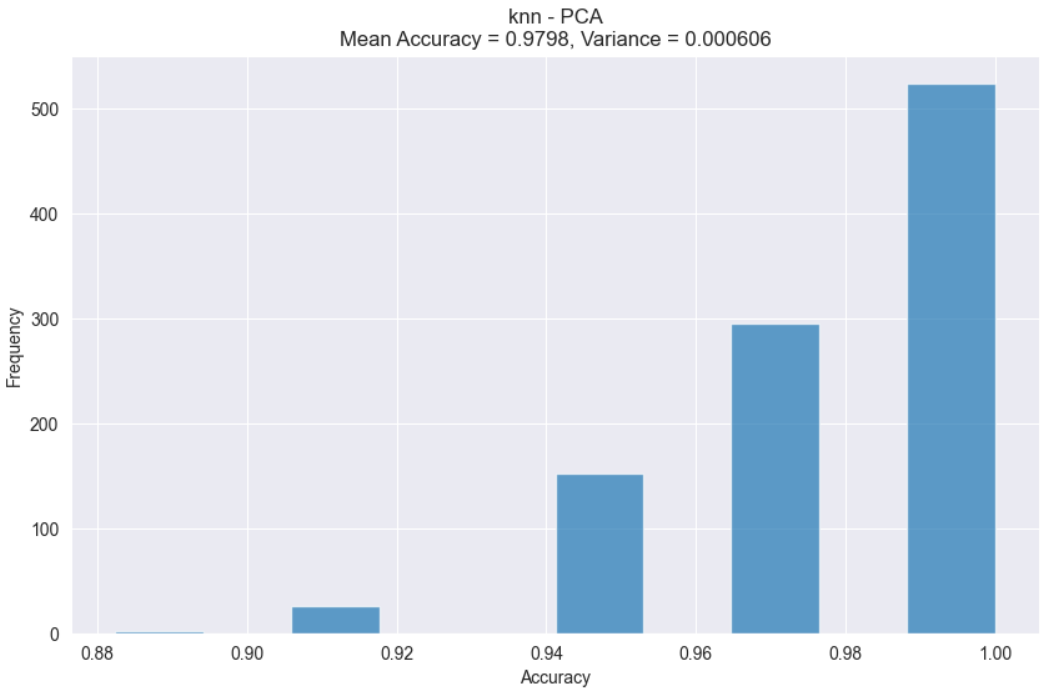


Figure 6: KNN

Metric	Logistic Regression	Naive Bayes	KNN
Mean Accuracy	0.9797	0.9766	0.9798
Variance	0.000605	0.000561	0.000606

After reducing the dimension spaces to 2, it appears KNN has the highest mean accuracy with prediction for our dataset while Naive Bayes has the lowest variance, being more sure of its predictions over different data splits. The decision boundaries and data points are plotted below through the use of an sklearn implementation to display model decision boundaries.

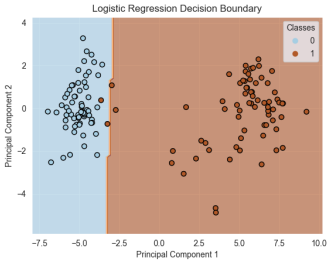


Figure 7: Logistic Regression

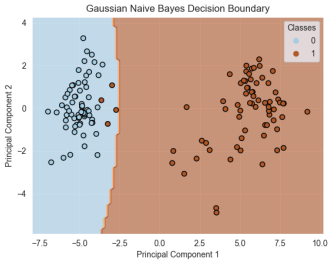


Figure 8: Naive Bayes

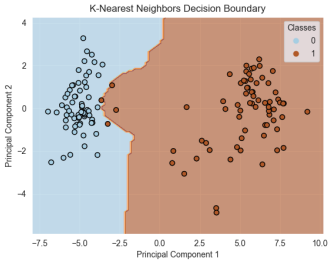


Figure 9: K-Nearest Neighbors

The decision for logistic regression appears to be a straight line indicating a linear decision boundary. Within 2 dimensions, this is a line, and in higher dimensions, a hyperplane. The decision boundary for Naive Bayes appears to be curved as it assumes Gaussian distribution and calculated the probability

of each data point belonging to a class. K-Nearest Neighbors is non-parametric and uses the distance between data points to classify them. A majority vote of the k nearest neighbors (in this case 5) determines which class a given point belongs to. As a result, the decision boundary for KNN can be quite irregular, sometimes even fragmented.

Sources

1. https://scikit-learn.org/dev/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
2. https://scikit-learn.org/dev/modules/generated/sklearn.inspection.DecisionBoundaryDisplay.html
3. https://scikit-learn.org/dev/modules/generated/sklearn.naive_bayes.GaussianNB.html
4. https://scikit-learn.org/1.5/modules/generated/sklearn.linear_model.LogisticRegression.html