Samuel Atkins
April 13, 2019
1002951754

ROB313 Assignmnent 5: Bayesian Inference

Assignment Objectives:

Learning Objectives:

1. To understand the fundamentals of Bayesian inference in the context of classification
2. To understand the impact of selecting higher and lower variances
3. To understand the advantages and disadvantages of using a Bayesian approach versus a frequentist approach
4. To learn about an up and coming topic in the machine learning field

Functional Objectives:

1. To fully implement a Bayesian inference model for logistic regression using a Bernoulli likelihood
2. To determine the best weights and the hessian of a classification dataset using the Laplace Approximation
3. To implement importance sampling to select a proposal distribution
4. To implement a Metropolis-Hastings MCMC sampler to estimate the posterior class
5. To review literature in the machine learning field

Code Structure Explanation:

The code is designed to be easy to follow. There are separate functions for each part of question 1. These functions take no arguments and automatically produce the desired plots and results. The following summarizes the most important functions contained in the BayesianInference.py file:

question1a()

Functionality:

- Approximates the log marginal likelihood of the following variances using the Laplace Approximation: [0.5, 1, 2]
  - Initializes the iris dataset, loops through each variance in the above variance list, and computes the marginal likelihood of each variance

Helper Functions Used:

- createMatrix(x)
  - Used to shape the x data into a matrix
- laplaceApproximation(x_train, x_test, y_train, y_test, learningRate, variance, maxIterations=inf)
  - Computes the Laplace Approximation of the supplied data using the supplied learning rate, variance, and max iteration count

- This function stops at a certain number of iterations. If it is not supplied maxIterations as an argument, the function stops looping when the gradient gets very small (i.e., < 0.000001)
    - In this function, a fixed amount of iterations were used to properly compare the marginal log likelihoods

Output:

- The marginal likelihood and iteration count for each variance

question1b()

Functionality:

- Chooses a proposal distribution and then implements importance sampling to estimate the most probable posterior class
- This function also visualizes the proposed distribution, estimates the most probable predictive posterior class on each element in the test set using a variance of 1, and then computes the test accuracy

Helper Functions Used:

- laplaceApproximation(x_train, x_test, y_train, y_test, learningRate, variance, maxIterations=inf)
    - Explained above
- multivariateNormal(mean, cov, size)
    - Samples randomly from a Gaussian distribution to produce a multi-dimensional normal distribution
- crossValidation(x_train, y_train, wList, H, variance), generateFolds(data)
    - Used to compute 5-fold cross validation over the training set to estimate the optimal weights
- posteriorPredictor(x_train, y_train, x_test, y_test, wProposed, qDistribution, variance)
    - Used to predict the posterior of the data by looping through all of the proposed weights

Output:

- A visualization of the data
- The results of the cross-validation process
- The test accuracy

question1c()

Functionality:

- Using a Metropolis-Hastings MCMC sampler, estimates the posterior class using a prior variance of 1 and proposed variances of [0.25, 0.5, 0.75, 1, 1.25, 1.5]
    - For every variance, burns in 1000 iterations, loops over 10000 iterations, and thins by collecting every 100th sample
    - Computes the weights using 5-fold cross-validation for each variance in the variance list

Helper Functions Used:

- laplaceApproximation(x_train, x_test, y_train, y_test, learningRate, variance, maxIterations=inf)

    - Explained above
    - Used to compute an initial estimate of the weights and the marginal likelihood
- MCMC(w, x_train, y_train, posteriorVariance, variance)
    - Used to compute w star given data, a prior variance, and a posterior variance
- crossValidationMCMC(x_folds, y_folds, wList, qDistribution, variance)
    - Used to compute an estimate for the weights using 5-fold cross validation (similar to the function used in question1b

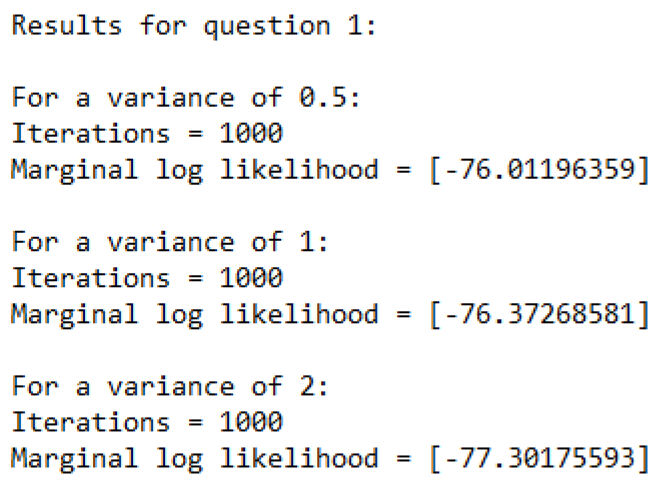
<u>Output</u>:

- For each proposed variance, displays the average accuracy of the cross-validation and then displays the optimal variance after the cross-validation stage.
- Using the optimal variance, performs classification on the test set and outputs the test accuracy along with the results of each test case
- Outputs a histogram of the 9th and 10th flowers

Question 1a:

In this question, the Laplace Approximation was used to compute the log marginal likelihood for the following variances: [0.5, 1, 2]. The training set and the validation set for the iris dataset were combined. The MAP solution was computed using gradient descent with a learning rate of 0.001. The hessian was computed at the MAP solution.

Given that the distribution of functions produced by a neural network will tend to a Gaussian process (Bishop), the greater the variance, the higher the model complexity. The marginal log likelihoods for the selected variances are displayed in Figure 1 below:



```
Results for question 1:

For a variance of 0.5:
Iterations = 1000
Marginal log likelihood = [-76.01196359]

For a variance of 1:
Iterations = 1000
Marginal log likelihood = [-76.37268581]

For a variance of 2:
Iterations = 1000
Marginal log likelihood = [-77.30175593]
```

Figure 1: Marginal log likelihood for each variance in the variance list

The largest log marginal likelihood, i.e. the marginal log likelihood of -76.012 retrieved by a variance of 0.5, suggests a model that could be too simple. The marginal log likelihood of -77.302 yielded by a variance of 2 is the most complex model and it may overfit. The model with a variance of 2 is the most complex model because it has the most negative marginal log-likelihood. We select a variance of 1 because it is in between the two models because this model likely will not over or underfit.

Question 1b:

In this question a proposal distribution was selected using the “numpy.random.multivariate_normal” import. This import samples from a Gaussian distribution randomly to generate a multi-dimensional normal distribution. The proposed distribution was formed using the weight vector calculated in question 1a as the mean and the negative Hessian calculated in question 1a as the covariance matrix. A code snippet of the calling of this function is shown below:

```
# Calculating proposed w using the multivariateNormal function:
wProposed = multivariateNormal(mean=w, cov=np.linalg.inv(-H)*variance, size=10000)
# Using a q distribution as proposed:
qDistribution = multivariate_normal.pdf(wProposed, mean=w, \
        cov=np.linalg.inv(-H)*variance)
```

The proposed and posterior distributions are plotted in the Figure below:

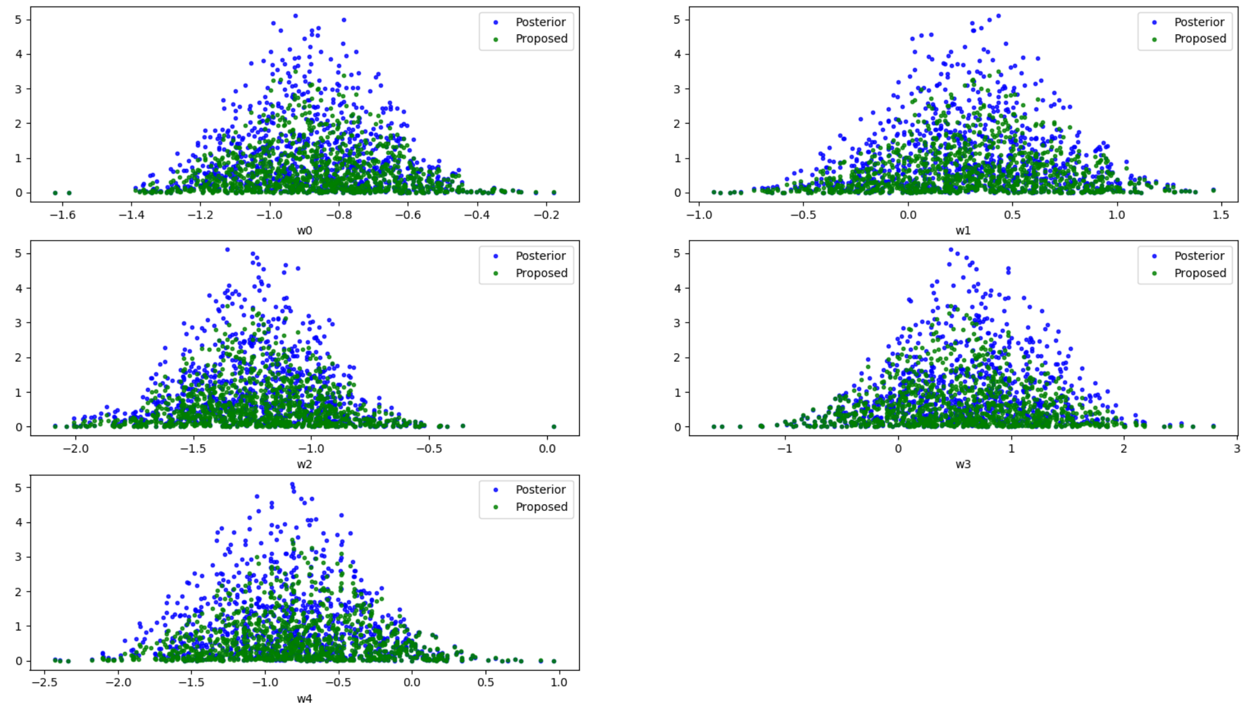
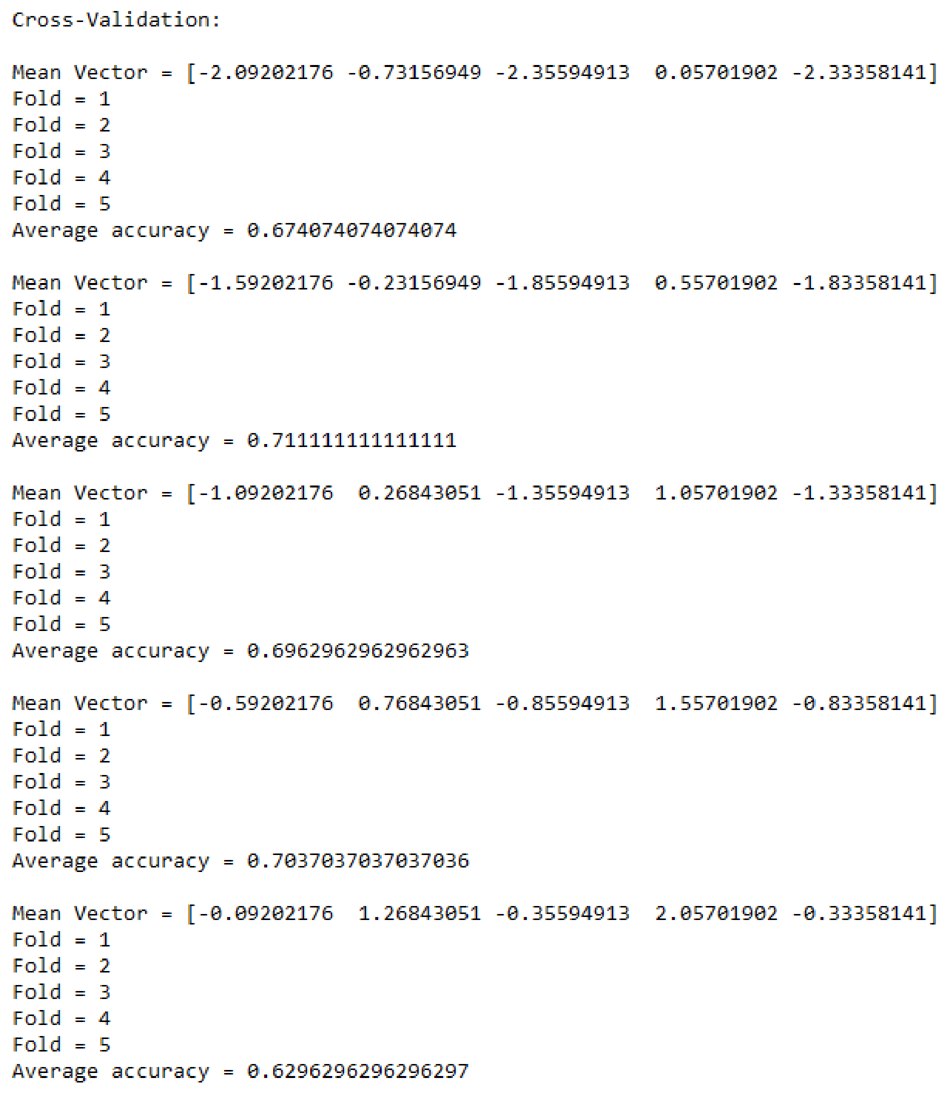


Figure 2: Proposed and posterior distributions

From the above figures we can see that the proposed distributions are very similar to the posterior distributions. The proposed distributions have more points deviating further away from the center, however, indicating that the tails of the proposed distributions are larger. This is a desired quality of a proposal distribution. Furthermore, we can see that there is significant overlap between the posterior and the proposal in areas of high probability mass. This is, once again, a desired quality of a proposal distribution.

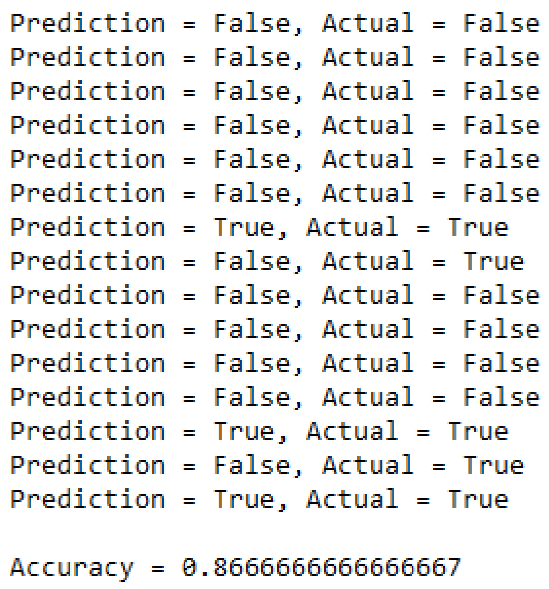
After selecting a distribution proposal, the optimal weight vectors were calculated. Five possible mean values were tested: [w – 1, w – 0.5, w, w + 0.5, w + 1]. These mean values were tested using 5-fold cross validation. Each mean was passed into the multivariate_normal function to produce a distribution. This proposed distribution was then evaluated using training set. The results of the cross-validation were outputted and are illustrated by Figure 3 below:



```
Cross-Validation:

Mean Vector = [-2.09202176 -0.73156949 -2.35594913  0.05701902 -2.33358141]
Fold = 1
Fold = 2
Fold = 3
Fold = 4
Fold = 5
Average accuracy = 0.674074074074074

Mean Vector = [-1.59202176 -0.23156949 -1.85594913  0.55701902 -1.83358141]
Fold = 1
Fold = 2
Fold = 3
Fold = 4
Fold = 5
Average accuracy = 0.711111111111111

Mean Vector = [-1.09202176  0.26843051 -1.35594913  1.05701902 -1.33358141]
Fold = 1
Fold = 2
Fold = 3
Fold = 4
Fold = 5
Average accuracy = 0.6962962962962963

Mean Vector = [-0.59202176  0.76843051 -0.85594913  1.55701902 -0.83358141]
Fold = 1
Fold = 2
Fold = 3
Fold = 4
Fold = 5
Average accuracy = 0.7037037037037036

Mean Vector = [-0.09202176  1.26843051 -0.35594913  2.05701902 -0.33358141]
Fold = 1
Fold = 2
Fold = 3
Fold = 4
Fold = 5
Average accuracy = 0.6296296296296297
```

Figure 3: Cross-validation of optimal weight calculation by varying the means of the multivariate normal distribution

From the above Figure, it is apparent that the proposed weight vector pertaining to a mean vector of [-1.59202176 -0.23156949 -1.85594913 0.55701902 -1.83358141] produced the greatest accuracy (71.1%). The proposed model then acted on the test set. The results of applying the optimal weight vector to the test set are shown in Figure 4 on the following page:



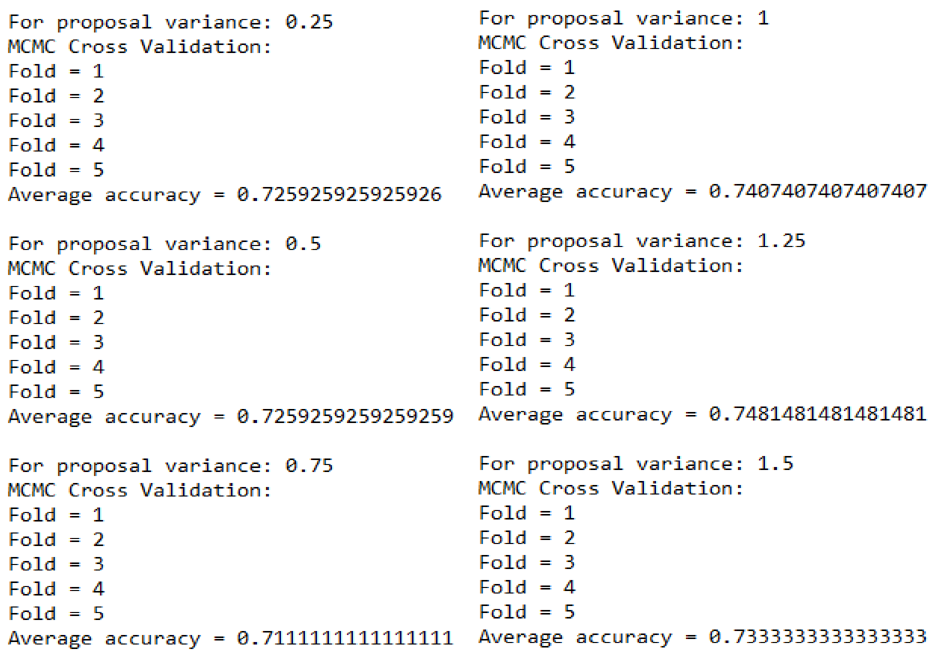
```
Prediction = False, Actual = False
Prediction = False, Actual = False
Prediction = False, Actual = False
Prediction = False, Actual = False
Prediction = False, Actual = False
Prediction = False, Actual = False
Prediction = True, Actual = True
Prediction = False, Actual = True
Prediction = False, Actual = False
Prediction = False, Actual = False
Prediction = False, Actual = False
Prediction = False, Actual = False
Prediction = True, Actual = True
Prediction = False, Actual = True
Prediction = True, Actual = True

Accuracy = 0.8666666666666667
```

Figure 4: Test set results using the optimal weight vector

From the above figure we can see that the test accuracy is 86.7%. This accuracy indicates that the proposal distribution that was selected was effective in classifying our data.

Question 1c:

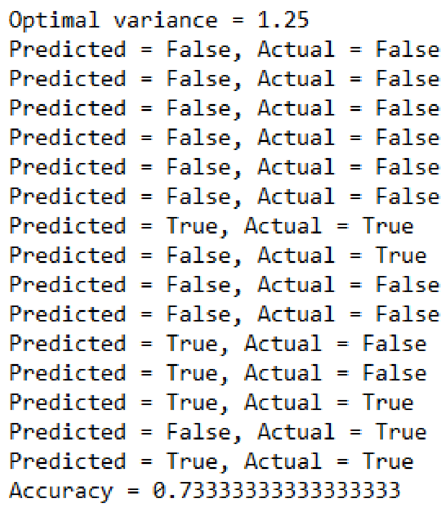
This question includes an MCMC Metropolis-Hastings sampler used to approximate the predictive posterior class on each element in the test set. A prior variance of 1 was used. The following variance values were selected for 5-fold cross-validation evaluation: [0.25, 0.5, 0.75, 1, 1.25, 1.5]. The results of the cross-validation are illustrated in the Figure below:



```
For proposal variance: 0.25                For proposal variance: 1
MCMC Cross Validation:                     MCMC Cross Validation:
Fold = 1                                   Fold = 1
Fold = 2                                   Fold = 2
Fold = 3                                   Fold = 3
Fold = 4                                   Fold = 4
Fold = 5                                   Fold = 5
Average accuracy = 0.725925925925926       Average accuracy = 0.7407407407407407

For proposal variance: 0.5                 For proposal variance: 1.25
MCMC Cross Validation:                     MCMC Cross Validation:
Fold = 1                                   Fold = 1
Fold = 2                                   Fold = 2
Fold = 3                                   Fold = 3
Fold = 4                                   Fold = 4
Fold = 5                                   Fold = 5
Average accuracy = 0.7259259259259259      Average accuracy = 0.7481481481481481

For proposal variance: 0.75                For proposal variance: 1.5
MCMC Cross Validation:                     MCMC Cross Validation:
Fold = 1                                   Fold = 1
Fold = 2                                   Fold = 2
Fold = 3                                   Fold = 3
Fold = 4                                   Fold = 4
Fold = 5                                   Fold = 5
Average accuracy = 0.7111111111111111      Average accuracy = 0.7333333333333333
```

Figure 5: Results of cross-validation for MCMC considering a variety of variances

From the results shown in the Figure above, we can see that the optimal variance is 1.25. Using this variance, the accuracy of the model was tested using the test set. These results are shown in Figure 6:



```
Optimal variance = 1.25
Predicted = False, Actual = False
Predicted = False, Actual = False
Predicted = False, Actual = False
Predicted = False, Actual = False
Predicted = False, Actual = False
Predicted = False, Actual = False
Predicted = True, Actual = True
Predicted = False, Actual = True
Predicted = False, Actual = False
Predicted = False, Actual = False
Predicted = True, Actual = False
Predicted = True, Actual = False
Predicted = True, Actual = True
Predicted = False, Actual = True
Predicted = True, Actual = True
Accuracy = 0.7333333333333333
```

Figure 6: Test results using an optimal variance of 1.25

We can see that the test accuracy for this model was 73.3% (11/15).

Figure 7 below illustrates a histogram of the 9th and 10th flowers:

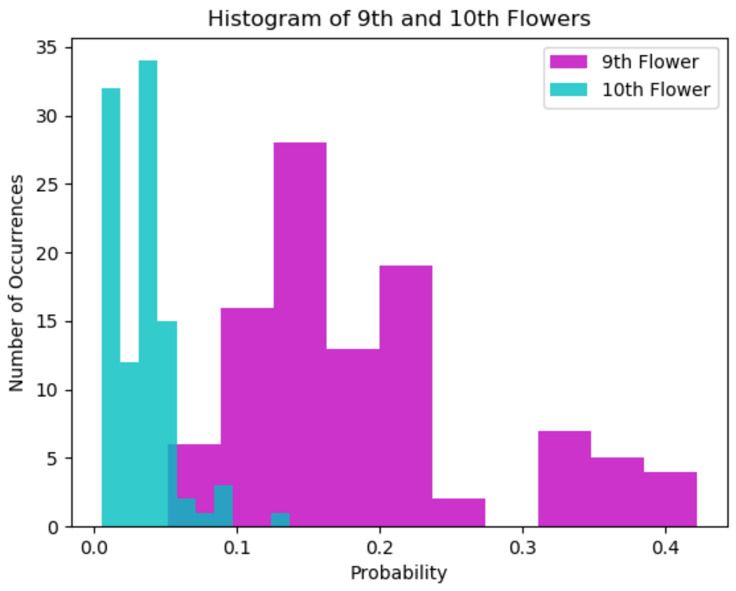


Figure 7: Histogram of the 9th and 10th flowers

The probability for the 10th flower favors the left whereas the 9th flower favors the middle/right more. Both the 9th and 10th flowers are truly false. Therefore, this plot makes sense because the distribution for both flowers is less than 0.5, indicating false.

Bayesian models use priors. With the use of priors Bayesian analysis informs us how likely something is to be the winner and indicates by how much. Frequentist models, on the other hand, do not use priors. These models often rely on thresholds and provide significantly less information. In the case of the 9th and 10th flowers illustrated above, a frequentist model would successfully categorize both flowers as false. For a flower that produced a Bayesian distribution around 0.5, however, a frequentist approach could likely incorrectly categorize it. A Bayesian approach, on the other hand, would present a distribution perhaps leaning a little more to one side than the other, providing us with more information.

Question 2:

The following is a report of the article titled “Engineering Safety in Machine Learning” by Kush R. Varshney of the IBM Thomas J. Watson Research Center in New York.

High-Level Overview:

Varshney introduces the article by proposing a broad definition of safety. He suggests that safety can best be defined as the minimization of knowledge uncertainty and risk for the purpose of the restricting unwanted outcomes. He then implies that the most logical next step is to investigate fields that do not have significant safety regulations in place such as machine learning. With this foundation set, he defines two possible machine learning application types. Type A applications affect other people’s lives profoundly whereas Type B applications are low consequence. After defining the two types of applications, Varshney discusses a few strategies to achieve safety as previously defined.

Summary:

Varshney begins the body of his article by briefly introducing some machine learning epistemic uncertainty concerns. He then expands on his definition of type A and B applications. He presents several examples of type A applications that have a tremendous effect on the people involved. He illustrates a scenario in which machine learning is used to determine if someone should be approved for a loan. He hints at other applications such as medical diagnosis and prison sentencing. He then refines his definition of type B applications by asserting that type B applications involve large amounts of data. He then suggests that because of this excess amount of data, there are typically significantly less errors in type B applications. Given that type A applications have a more severe impact on people’s lives and are more prone to error, Varshney asserts that these applications are of particular interest. This classification of machine learning applications is captivating because it implores the reader to consider the less abstract side of machine learning. The concept of machine learning can be foreign to many. Providing real-world examples of impactful machine learning problems brings the topic closer to home.

With that, he begins to discuss a few strategies for achieving safety. He discusses 4 possible strategies. Firstly, he discusses inherently safe design. This strategy consists of excluding the possibility of hazards from the system. In the context of machine learning this strategy is equivalent to pruning the dataset and ensuring that the system is trained with the appropriate knowledge of the possible biases of the data. The second strategy consists of ensuring additive reserves are present. In the context of machine learning, this means parameterizing the unknown to ensure that risk is accounted for in the worst case. The third strategy is introducing safe fails. Varshney presents an electrical fuse on a circuit board as an example of a safe fail unrelated to machine learning would. In the context of machine learning, an example of a safe fail is the reject option. I find the electrical fuse analogy especially powerful. Creating a fail safe seems completely necessary when dealing with electrical circuits but seems less important with respect to machine learning. An example of a fail-safe in the context of machine learning is instead of categorizing something with low confidence, a model with this type of fail-safe refuses to categorize at all. The final strategy introduced by Varshney is called procedural safeguarding. This strategy, with respect to machine learning, is implemented when the machine learning system is open source. This prevents incorrect classification and irresponsible system configuration, thus, increasing safety. The safety strategies presented in this article were presented in a very profound way.