CSE 512: Homework 2 Solutions Due September 22nd 2021

1. **(2 pts) Bayes vs 1NN classifier.** [1] I'd like to collect some Pokemon of which to do battle. Without knowing much, and my primary criteria is to find Pokemon who can throw things to a very far distance. In particular, I will only accept Pokemon who can throw trajectories at least 72 inches (6 ft) distance.

   In general, if a Pokemon says he/she can throw x inches, then in truth they are lying, and their true ability follows a Gaussian distribution with mean x and variance $\sigma^2 = 6^2 = 36$ inches.

   For parts (a) and (b), you may use `wolframalpha.com` to help compute the integral. Report both the symbolic form and the computed number up to 3 significant digits. (We will first try to match your numbers, and if your numbers are incorrect, we will use the symbolic form for partial credit.)

   (a) **(0.5 pts)** Arbok claims that he can throw 70 inches. What is the probability that he fulfills my criteria?

   (b) **(0.5 pts)** Bulbasaur claims he can throw 60 inches. What is the probability that he is can actually throw further than Arbok? (Assume their throwing arms are independent.)

   (c) Charmander is the next Pokemon who crosses my path, and he claims to throw exactly 72 inches. Consider the following reward function: If I accept him, and Charmander does fulfill this criteria, then he wins tons of battles and I get +10 reward. If he in fact is not fulfilling my critiera, I waste my resources, he loses battles, and I end up with a deadbeat Pokemon, translating to a -1 reward. If I reject Charmander and in fact was able to throw far enough, then someone else will scoop him up and destroy my other Pokemon, giving me a -25 reward. But if I reject Charmander and he was in fact unable, there is 0 reward.

   Think of loss as negative reward.

   i. **(0.4 pts)** What is the Bayes risk of accepting vs rejecting?

   ii. **(0.1 pts)** What does the Bayes classifier tell me to do?

   (d) **(0.5 pts)** Diglett, Eevee, and Flareon are former Pokemons of mine, who have all lied about their abilities. Diglett wrote he throws 85 inches but actually he can only throw 60 inches. Eevee wrote she throws 72 inches and in fact she throws closer to 70 inches. Flareon wrote 90 inches and it's not a bad estimate; he throws 89 inches. Think of these three Pokefriends as my training dataset.

   Gardevoir suddenly appears and walks across my keyboard. I ask him "How far can you throw your projectiles?" and he sniffs, and says "82 inches, easily." Does a 1-nearest neighbor regressor predict that Gardevoir fulfills my critiria? Explain your answer.

2. **(3 pts) Decision theory.** I run a factory that makes widgets and gadgets. Despite best efforts, manufacturing defects can always occur. I would like to inspect each of these items individually, but the cost of inspection is pretty high, so I cannot inspect each individual widget and gadget.

   The widgets and gadgets are printed on disks. A disk has a 10% chance of being warped. There are two printing presses, a blue one and a red one. The table below gives the possibility that, given a disk of a particular state printed by a particular press, a widget or gadget printed on that disk is warped.

disk \ press	red	blue
warped	30%	85%
normal	5%	0 %

   (To interpret the table, the probability that a gadget is defective if it were on a disk that is not warped, and printed by a red press, is 5%.)

   (a) First, we consider only the loss of quality in a product. That is, if we ship a widget or gadget is defective, we incur a loss of +1. Otherwise, we incur no losses.

[1] All numbers in this problem are made up.

i. **(0.5 pts)** Without inspecting anything (that is, we ship out everything we make), what is the Bayes risk of using a red press, over a single disk? a blue press? Which machine would I use to minimize the Bayes' risk?

ii. **(0.2 pts)** Without inspecting anything, what is the Minimax risk of using a red press? a blue press?

iii. **(0.2 pts)** Suppose I invest the effort into inspecting disks, and remove all warped disks. What is the Bayes risk, per disk, of using a red press? a blue press?

iv. **(0.1 pts)** After removing all warped disks, what is the minimax risk of using a red press? a blue press?

(b) Widgets are primarily used in online advertising. If they are defective, they will end up sending an ad that is undesirable. However, if they are removed, then no ad is sent out. Therefore, the revenue gained from a widget is estimated at

$$\text{revenue per widget} = \begin{cases} \$1 & \text{if widget is sold and is not defective} \\ -\$1 & \text{if widget is sold and is defective} \\ 0 & \text{if the widget is not sold.} \end{cases}$$

There is no cost to rejecting a disk, but the cost of inspection is \$1 per percent of disks inspected, per widget. (So, if I inspect every disk, I pay \$100 per widget. If I inspect only 10% of disks, I pay \$10 per widget.) Every widget that is not on a disk that was found to be warped is sold.

i. **(0.5 pts)** Compute the Bayes reward (e.g. the expected profit per day) as a function of $x = \mathbf{Pr}(\text{inspection})$ for widgets, when using the blue press. Compute the same for the red press. (Remember that we are computing the cost *per widget*.)

ii. **(0.5 pts)** If you were a consultant for my factory, how much inspection would you recommend? Would you recommend using one press over the other for widgets?

(c) Gadgets are primarily used in medical care. If they are defective, someone will die. However, if they are removed, then someone waits a day longer to get a much-needed test. While we can never assign monetary value to a human life, in terms of insurance costs experts have estimated the following value:

$$\text{revenue per gadget} = \begin{cases} \$500 & \text{if gadget is sold and is not defective} \\ -\$10,000 & \text{if gadget is sold and is defective} \\ \$0 & \text{if the gadget is not sold.} \end{cases}$$

Again, there is no cost to rejecting a disk, and again, the cost of inspection is \$1 per percent of disks inspected, per gadget. Every gadget that is not on a disk that was found to be warped is sold.

i. **(0.5 pts)** Compute the Bayes reward (e.g. the expected profit per day) as a function of $x = \mathbf{Pr}(\text{inspection})$ for gadgets, when using the blue press. Compute the same for the red press.

ii. **(0.5 pts)** If you were a consultant for my factory, how much inspection would you recommend? Would you recommend using one press over the other for gadgets?

3. **(3 pts)** ***K*-nearest neighbors classification.** We will now try to use the KNN classifier to classify MNIST digits.

- Open `hw2_minst_release.ipynb`. Load the necessary packages and the data, and take a look at how the data is formatted and structured. I have done all the "data cleaning" needed for this assignment (which is very minimal for this exercise). I have also included a function `get_small_dataset` which will return a subset of the training data (60000 samples!) so that we can reasonably train some things on even the worst laptops.
- **(0.5 pt) Distance function.** The first step in establishing a KNN classifier is deciding what is going to be your metric for "distance", and writing a function that, given the training data `Xtrain` and query point `zquery`, can as efficiently as possible return a vector of distances between `zquery` and all of the datapoints in `Xtrain`.

  There are many ways to do this, some faster than others. In general, if your implementation involves a `for loop`, you may be in for a lot of waiting and some very warm laptops. One implementation that avoids `for loops` is to really try to use the optimized numerical linear algebra functions of the numpy library as much as possible, e.g. using functions like `np.dot`, `np.sum`, etc.

The distance function we will use is the 2 norm squared. It may help to see this metric expanded, e.g.

$$d(x_i, z) = \|x_i - z\|_2^2 = x_i^T x_i - 2x_i^T z + z^T z.$$

One "optimized" approach is to compute each term separately, using optimized numpy functions. (Although the code is not set up this way, you could even further optimize things by computing the terms involving only the training data ahead of time.)

In your writeup, print what you see when you run the box, e.g. the print outputs of

```
print(get_dist(Xtrain,Xtrain[0,:])[0])
print(get_dist(Xtrain,Xtest[0,:])[10])
print(get_dist(Xtrain,Xtest[10,:])[50])
```

- **(1 pts) Prediction.** Implement a K-nearest-neighbor classification predictor, which takes a test data point, finds the closest point (in terms of Euclidean distance) in the train data set, and returns the KNN prediction. Use a majority vote scheme to decide which label to return; use whatever scheme you wish to break ties.

  Hint: take a look at `scipy.stats.mode()`

  In your writeup, print the output for $K = 3$ and $m = 100$, e.g. the output for the lines

  ```
  print(ytest_pred[:20])
  print(ytest[:20])
  ```

- **Evaluate based on classification accuracy.** Now write a function that returns the classification accuracy given a list of true labels (`ytrue`) and predicted labels (`ypred`).

  In your writeup, print classification accuracy of the test set.

- **(1 pts) Hyperparameter tuning.** I have included in the next box an experiment in which your KNN predictor is tested for a training dataset with 100, 1000, and 2500 data samples. In each case, the code will run your predictor and return three numbers: m, the prediction accuracy, and the prediction runtime. Run this box and **return the classification accuracy and runtimes** for $m = 100, 1000, 2500$ and $K = 1, 3, 5$.

m	K	accuracy	runtime (sec)
100	1		
100	3		
100	5		
1000	1		
1000	3		
1000	5		
2500	1		
2500	3		
2500	5		

  Comment a bit on the performance of the model for these different hyperparameter choices. In particular:

  - Is it feasible to run this model for the full $m = 60000$ training dataset in runtime? Is it advisable?
  - How does the accuracy depend on K for different values of m?

4. **(2 pts) Naive Bayes and Alice in Wonderland.**

- Open the python notebook `hw2_alice_naivebayes_release.ipynb` . After running the first couple boxes, you should have loaded the entire text of "Alice in Wonderland" by Lewis Caroll, as an ordered list of words. Our task today will be to do word prediction based on this corpus. Throughout this exercise, this corpus will serve as both our training and testing data.
- **Tokenize** While the exact word means a lot to us, for a (primitive) computer, a word is just some object; in particular, we represent each unique word as a unique number. This is the word's token. Run the 3rd block to tokenize the data, and understand what it is doing.

(a) **(0.5 pt) Bigram classifier**. We predict the next word using only the previous word. Here, we should think of features x as the previous word, and label y as the next word. Therefore, a fully populated table of $\mathbf{Pr}(x|y)$ should have $V \times V$ entries, where V is the size of the vocabulary.

Populate the next box with the calculation for the posterior $\mathbf{Pr}(x|y)$ and the prior $\mathbf{Pr}(y)$ based on the statistics of the corpus. Do not worry about normalization, e.g. the likelihood function can return the first term of

$$\mathbf{Pr}(x|y)\mathbf{Pr}(y) \propto \mathbf{Pr}(y|x).$$

Now construct a Bayes classifier using only this feature. **Report the classification accuracy** over the entire corpus of this classifier.

(b) **(0.5 pt)** Well, that was pretty terrible. Let's try and incorporate not just the past word, but the past 2 words. Fill in the posterior for $\mathbf{Pr}(x|y)$ where now x is the 2nd previous word. Now construct a Naive Bayes classifier that uses both the past and past 2nd word as features. **Report the classification accuracy** over the entire corpus of this classifier. (Note that this is not a 3-gram classifier, which would be the not-Naive-Bayes version of what we are doing here.)

- **(0.25 pt)Text generation** Using the likelihoods computed from the bigram classifier, and starting with a seed word "alice", generate the next 25 words by always picking the most likely next word.
- **(0.25 pt)Text generation** Do the same thing with the 2-past-words Naive Bayes likelihood, starting with the seed phrase "alice was".
- **(0.5 pt, 0.25 per generation)** In both of those cases, you should have found that the generator was terrible, and gets stuck in a loop through a very short and uninteresting phrase. We could try to spice things up a little, however. In your generation script, instead of returning the next word with highest likelihood, sample the next word according to the likelihood weights. (hint: check out `random.choices()`) Are either of these good generators, in your view?

Challenge!

(2 pts) Today we have two separate questions, each worth one point.

1. The n-gram version of the classifier from the Alice in Wonderland exercise would be the not-Naive-Bayes version; that is, rather than just counting how many times "mad" precedes "hatter" and "the" is 2 words previous of "hatter", we would like to count exact phrases, e.g. how many times "the mad" precedes "hatter". If we count 1 word ago, we are building a bigram classifier; 2 words is trigram, etc. In general, an n-gram classifier looks for the probabilities of all n-grams, e.g. given a history of $n-1$ words, predict the next best word.

   In this exercise, build an n-gram classifier for "Alice in Wonderland". Note that using a full memory of every possible combination of n words will quickly become memory inefficient, so you want to "leverage sparsity"; that is, only store the phrases you see. Then, you will find that you can extend n quite a bit with no problem.

   What to do:

   - Using the n-gram classifier trained on "Alice in Wonderland", **give the train accuracy** (on same corpus) and **test accuracy** (on Alice through the looking glass) for next-word-prediction.
   - Sweep the hyperparameters for $n = 2, 3, ..., 100$, and give the train and test accuracy for each value. Do you notice a "best" value of n? At what values of n do you detect overfitting, e.g. train accuracy $\gg$ test accuracy?
   - For a good choice of hyperparameter, show the result of 100 words generated using any seed found in either corpus. Do this by sampling the likelihood randomly. Is this new generator any good, in your opinion?

2. **Linearly hashed KNN** If your computer is anything like my dinky laptop, that KNN exercise was a pain to run. As we discussed in class, in general, KNN is tough on runtime memory and computation, because in general we don't like doing things like training during prediction. One way to get around this is **dimensionality reduction**. While there are many clever ways of implementing such a thing, for this exercise we will just try something very simple, yet sometimes shockingly effective: **linear hashing**.

   Linear hashing basically follows the principle (sometimes formally stated as the Johnson-Lindenstrauss lemma):

   $$\mathbf{dist}(x, y) \approx \mathbf{dist}(Ax, Ay)$$

   where $A \in \mathbb{R}^{n \times d}$ is some random matrix with $d \ll n$. That is to say, if I take some points in space, and hit them with a random projection to a lower dimensional space, their configuration should be roughly preserved. We won't really go into this until later lectures, but for now we can evaluate this principle empirically, by creating linearly hashed KNN.

   Add a new box in your KNN ipython notebook. Code in two hashing functions:

   - **Random subsampling** Pick a random subset $\mathcal{S} \subset \{1, ..., n\}$, $|\mathcal{S}| = d$ and reduce each feature vector so that only the features in $\mathcal{S}$ remain.
   - **Random matrix multiply** Generate a random matrix $A \in \mathbb{R}^{d \times n}$ where each A_{ij} is i.i.d. generated according to $\mathcal{N}(0, 1)$. Then the new feature is $z = Ax$.

   Code up KNN over the *hashed* vectors. Make sure you also hash vectors before prediction. Return the classification accuracy and runtimes for $m = 100, 1000, 10000$. What are your thoughts on using hashing for KNN? In particular:

   - Does the reduction in runtime justify the accuracy hit? In what scenarios would we prefer to use hashing vs not hashing?
   - Comment on the performance difference between hashing via subsampling and hashing via random matrices.