Name: Ya Zou
Date: 5 Oct 2023

CS434 Machine Learning and Data Mining – Homework 1

k-Nearest Neighbor Classification and Statistical Estimation

1 Statistical Estimation [10pts]

The Poisson distribution is a discrete probability distribution over the natural numbers starting from zero (i.e. 0,1,2,3,...). Specifically, it models how many occurrences of an event will happen within a fixed interval. For example – how many people will call into a call center within a given hour. Importantly, it assumes that an individual event (a person calling the center) is independent and occurs with a fixed probability. Say there is a 2% chance of someone calling every minute, then on average we would expect 1.2 people per hour – but sometimes there would be more and sometimes fewer. The Poisson distribution models that uncertainty over the total number of events in an interval.

The probability mass function for the Poisson with parameter λ is given as:

$$\texttt{Pois}(X = x; \lambda) = \frac{\lambda^x e^{-\lambda}}{x!} \qquad \forall x \in \{0, 1, 2, ...\} \qquad \lambda \geq 0 \tag{1}$$

where the rate λ can be interpreted as the average number of occurrences in an interval. In this section, we'll derive the maximum likelihood estimate for λ and show that the Gamma distribution is a conjugate prior to the Poisson.

▶ Q1 Maximum Likelihood Estimation of λ [4pts]. Assume we observe a dataset of occurrence counts $D = \{x_1, x_2, ..., x_N\}$ coming from N i.i.d random variables distributed according to $\texttt{Pois}(X = x; \lambda)$. Derive the maximum likelihood estimate of the rate parameter λ. To help guide you, consider the following steps:

1. Write out the log-likelihood function $logP(D|\lambda)$.
2. Take the derivative of the log-likelihood with respect to the parameter λ.
3. Set the derivative equal to zero and solve for λ – call this maximizing value $\hat{\lambda}_{MLE}$.

Your answer should make intuitive sense given our interpretation of λ as the average number of occurrences.

1) The PMF for the Poisson is given

$$\texttt{Pois}(X = x; \lambda) = \frac{\lambda^x e^{-\lambda}}{x!} \qquad \forall x \in \{0, 1, 2, ...\} \qquad \lambda \geq 0$$

Therefore if we want to find the probability of our observed data given a parameter lamda, we can multiply the probability:

λ

$$P(X = x \mid \lambda) = \prod_{i=1}^{n} \frac{\lambda^{x_i} * e^{\lambda}}{x_i!}$$

We can apply the log function, which will not change the lambda value which corresponds to the maximum liklihood, thus we can get:

$$\ln P(D|\lambda) = \ln\left(\prod_{i=1}^{n} \frac{\lambda^{x_i} * e^{-\lambda}}{x_i!}\right)$$

$$\Rightarrow \ln P(D|\lambda) = \sum_{i=1}^{n} \ln\left(\frac{\lambda^{x_i} * e^{-\lambda}}{x_i!}\right)$$

$$\Rightarrow \ln P(D|\lambda) = \sum_{i=1}^{n} \left(\ln(\lambda^{x_i} * e^{-\lambda}) - \ln(x_i!)\right)$$

$$\Rightarrow \ln P(D|\lambda) = \sum_{i=1}^{n} \left(\ln(\lambda^{x_i}) + \ln(e^{-\lambda}) - \ln(x_i!)\right)$$

$$\Rightarrow \ln P(D|\lambda) = \sum_{i=1}^{n} \left(x_i * \ln(\lambda) + (-\lambda * \ln(e)) - \ln(x_i!)\right)$$

$$\Rightarrow \ln P(D|\lambda) = \sum_{i=1}^{n} \left(x_i * \ln(\lambda) + (-\lambda * 1) - \ln(x_i!)\right)$$

$$\Rightarrow \ln P(D|\lambda) = \sum_{i=1}^{n} \left(x_i * \ln(\lambda) - \lambda - \ln(x_i!)\right)$$

2)&3) we are going to take drive of the equation (log-likelihood) we just got above, then we will set the derivative equation equals to 0.

$$\frac{\partial}{\partial \lambda} \ln P(D|\lambda) = \frac{d}{d\lambda}\left(\sum_{i=1}^{n} (x_i * \ln(\lambda) - \lambda - \ln(x_i!))\right) = 0$$

$$\Rightarrow \sum_{i=1}^{n} \frac{d}{d\lambda}\left((x_i * \ln(\lambda) - \lambda - \ln(x_i!))\right) = 0$$

$$\Rightarrow \sum_{i=1}^{n} \frac{d}{d\lambda}\left(x_i * \ln(\lambda) - \frac{d}{d\lambda}(\lambda) - \frac{d}{d\lambda}(\ln(x_i!)) = 0 \right.$$

$$\Rightarrow \sum_{i=1}^{n} \left(x_i * \frac{d}{d\lambda}(\ln(\lambda)) - 1 - 0 \right) = 0$$

$$\Rightarrow \sum_{i=1}^{n} \left(\frac{x_i}{\lambda} - 1 \right) = 0 \quad \Rightarrow \frac{\sum_{i=1}^{n} x_i}{\lambda} = n$$

$$\Rightarrow \hat{\lambda} = \frac{\sum_{i=1}^{n} x_i}{n} \quad \Rightarrow \hat{\lambda} = \bar{x}$$

Thus. we can observe that λ as average number of occurance

▶ Q2 Maximum A Posteriori Estimate of λ with a Gamma Prior [4pts]. As before, assume we observe a dataset of occurrence counts $D = \{x_1, x_2, ..., x_N\}$ coming from N i.i.d random variables distributed according to $\texttt{Pois}(X = x; \lambda)$. Further, assume that λ is distributed according to a $\texttt{Gamma}(\Lambda = \lambda; \alpha, \beta)$. Derive the MAP estimate of λ. To help guide you, consider the following steps:

1. Write out the log-posterior $logP(\lambda|D) \propto logP(D|\lambda) + logP(\lambda)$.
2. Take the derivative of $logP(D|\lambda) + logP(\lambda)$ with respect to the parameter λ.
3. Set the derivative equal to zero and solve for λ – call this maximizing value $\hat{\lambda}_{MAP}$.

1. According to Bayes' Rule we can get:

$\text{argmax}_\lambda(P(\lambda|D)) = \text{argmax}_\lambda \frac{(P(D|\lambda) * P(\lambda)}{P(D)}$ and we notice that $P(D)$ is not depend on λ, so we can take it out, thus we can get:

$\text{argmax}_\lambda(P(\lambda|D)) = \text{argmax}_\lambda(P(D|\lambda) * P(\lambda))$.

The given Gamma distribution as below:

$$\text{Gamma}(\Lambda = \lambda; \alpha, \beta) = \frac{\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)}, \forall \lambda \geq 0 \quad \alpha, \beta > 0$$

then we can substitute into the previous equation then we can get:

$$\text{argmax}_\lambda\left(P(\lambda|D) = \text{argmax}_\lambda\left(\prod_{i=0}^{N}\left(\frac{\lambda^{x_i} * e^{-\lambda}}{x_i!}\right) * \frac{\beta^\alpha * \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)}\right)\right.$$

now we can apply log to the equation we get:

$$\text{argmax}_\lambda\left(\ln\left(P(\lambda|D)\right) = \text{argmax}_\lambda\left(\ln\left(\prod_{i=0}^{N}\left(\frac{\lambda^{x_i} * e^{-\lambda}}{x_i!}\right) * \frac{\beta^\alpha * \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)}\right)\right)\right.$$

$$= \text{argmax}_\lambda\left(\sum_{i=0}^{N}\ln\left(\frac{\lambda^{x_i} * e^{-\lambda}}{x_i!}\right) * \ln\left(\frac{\beta^\alpha * \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)}\right)\right)$$

2. now we can take drivitive of $\log(D|\lambda) + \log P(\lambda)$ with respect parameter λ and set drivitive equal to zero and solve for λ

$$\frac{d}{d\lambda}\left(\sum_{i=0}^{n}\ln\left(\frac{\lambda^{x_i} e^{-\lambda}}{x_i!}\right) * \ln\left(\frac{\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)}\right)\right)$$

$$\Rightarrow \frac{d}{d\lambda}\left(\sum_{i=10}^{n}\left(\ln\left(\lambda^{x_i} e^{-\lambda}\right) - \ln(x_i!) + \ln\left(\frac{\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)}\right)\right)\right.$$

$$\Rightarrow \frac{d}{d\lambda}\left(\sum_{i=10}^{n}\left(\ln(\lambda^{x_i}) + \ln(e^{-\lambda}) - \ln(x_i!) + \ln(\beta^\alpha * \lambda^{\alpha-1} * e^{-\beta\lambda}) - \ln(\Gamma(\alpha))\right)\right)$$

$$\Rightarrow \frac{d}{d\lambda}\left(\sum_{i=0}^{n}\left(\ln(\lambda^{x_i}) + \ln(e^{-\lambda}) + \ln(\lambda^{\alpha-1}) + \ln(e^{-\beta\lambda})\right)\right)$$

$$\Rightarrow \frac{d}{d\lambda}\left(\sum_{i=0}^{n}\left(x_i\ln(\lambda) - \lambda\ln(e) + (\alpha-1)\ln(\lambda) - \beta\lambda\ln(e)\right)\right)$$

$$\Rightarrow \frac{d}{d\lambda}\left(\sum_{i=0}^{n}\left(x_i\ln(\lambda) - \lambda\right) + (\alpha-1)\ln(\lambda) - \beta\lambda\right.$$

$$\Rightarrow \sum_{i=0}^{n} \left(x_i \frac{1}{\lambda} - 1 \right) + (\alpha - 1)\frac{1}{\lambda} - \beta$$

$$\Rightarrow \sum_{i=0}^{n} \left(\frac{x_i}{\lambda} - 1 \right) + \frac{\alpha - 1}{\lambda} - \beta$$

3. now we can set drivative equal to zero, and solove for λ to maximize $\hat{\lambda}$ MAP

assume $\lambda = x_i$, thus we can get

$$\sum_{i=0}^{n} \left(\frac{\lambda}{\lambda} - 1 \right) + \frac{\alpha - 1}{\lambda} - \beta = 0$$

$$\Rightarrow \frac{\alpha - 1}{\lambda} = \beta$$

$$\Rightarrow \lambda = (\alpha - 1)\beta$$

In the previous question we found the MAP estimate for λ under a Poisson-Gamma model; however, we didn't demonstrate that Gamma was a conjugate prior to Poisson – i.e. we did not show that the product of a Poisson likelihood and Gamma prior results in another Gamma distribution (or at least is proportional to one).

▶ Q3 Deriving the Posterior of a Poisson-Gamma Model [2pt]. Show that the Gamma distribution is a conjugate prior to the Poisson by deriving expressions for parameters α_P, β_P of a Gamma distribution such that $P(\lambda|D) \propto \texttt{Gamma}(\lambda; \alpha_P, \beta_P)$.

[Hint: Consider $P(D|\lambda)P(\lambda)$ and group like-terms/exponents. Try to massage the equation to looking like the numerator of a Gamma distribution. The denominator can be mostly ignored if it is constant with respect to λ as we are only trying to show a proportionality ($\propto$).]

Fist lets see the Poisson distribution as below:

$$P(X=x|\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$ and the Gamma distribution as following: $\text{Gamma}(\alpha, \beta) = \frac{\beta^\alpha \cdot \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)}$, $\lambda > 0$

we also know Bayes's Rule as following $P(\lambda|D) = \frac{P(D|\lambda)P(\lambda)}{P(D)}$

accordig to the given information, we can know that

$P(D|\lambda) = \prod_{i=1}^{n}\left(\frac{\lambda^{x_i} e^{-\lambda}}{x_i!}\right)$, also can be written as the following

$$P(D|\lambda) = \frac{\lambda^{x_i} e^{-\lambda}}{\prod_{i=1}^{n} x_i!}$$, we can notice that

$\prod_{i=1}^{n} x_i!$ is independent of λ, so we can write as

$P(D|\lambda) = \lambda^{n\hat{x}} e^{-n\lambda}$ Because $\sum_{i=1}^{n} x_i = n\hat{x}$. now we can

Put the equation we got back to the original equation as the following: $P(D|\lambda) = \lambda^{n\hat{x}} e^{-n\lambda} (\lambda^{\alpha-1} e^{-\beta\lambda})$ Because β^α and $\Gamma(\alpha)$ are independent of λ. Lets rearrange the equation $P(\lambda|D) = \lambda^{n\hat{x}+\alpha-1} * e^{-(n+\beta)\lambda}$

$\therefore \quad P(\lambda|D) \propto \text{Gamma}(\lambda; \alpha_p, \beta_p)$ where

$\alpha_p = \sum_{i=1}^{n} x_i + \alpha + 1 = n\hat{x} + \alpha - 1$ and $\beta_p = \beta + n$.

This show Gamma distribution is the conjugate prior to the Poisson.

▶ Q4 Encodings and Distance [2pt] To represent nominal attributes, we apply a one-hot encoding technique – transforming each possible value into its own binary attribute. For example, if we have an attribute *workclass* with three possible values Private, State-gov, Never-worked – we would binarize the workclass attribute as shown below (each row is a single example data point):

workclass
Private
State-gov
Never-worked
State-gov

The original field

workclass= Private	workclass= State-gov	workclass= Never-worked
1	0	0
0	1	0
0	0	1
0	1	0

Fields after binarization

A common naive preprocessing is to treat all categoric variables as ordinal – assigning increasing integers to each possible value. For example, such an encoding would say 1=Private, 2=State-gov, and 3=Never-worked. Contrast these two encodings. Focus on how each choice affects Euclidean distance in kNN.

4. **Ordinal**:

Ordinal encoding assigns numeric values to categorical attributes, making them as if they have linear relationship, but these numeric value do not necessarily refelect the actual similarity between categories. The Euclidean distance calculated using ordinal encoding may not accurately represent the dissimilarity between data points.Fo instance

$$ED(\text{Private}, \text{Never-Worked}) = \sqrt{1^2 + 3^2} = 3.16$$

The number 3.16 doesn't mean anything here in this context, because there is no inherent meaning to the numbers 1,2,3. Even if there are some meaning behind these numbers, if we have 100 categories, the distance between the fist category and the last category will be very large.

One hot :

It creates a binary representation for categorical values, where each attribute is either 1 or 0, this means Euclidean distance between data points in the d-dimensional space create by the numbers of differences between the binary attributes, if the distance between them is larger, this can lead to a more accurate representation of dissimilarity between data points for categorical attributes. For example we calculate the same distance between private and never/ worked:

$$ED(\text{Private}, \text{Never-worked}) = \sqrt{(1-0)^2 + (0-0)^2 + (0-1)^2 + (0-0)^2} \approx 1.41$$

Thus, one-hot encoding provides more accurate representation of dissimislarity between data points. However, the ordinal encoding can misleading distance relationships, as it assumes a linear relationship that might not exist data.

▶ Q5 Looking at Data [3pt] What percent of the training data has an income >50k? Explain how this might affect your model and how you interpret the results. For instance, would you say a model that achieved 70% accuracy is a good or poor model? How many dimensions does each data point have (ignoring the id attribute and class label)? [Hint: check the data, one-hot encodings increased dimensionality]

$$\frac{1967 \text{ data points has an income } > 50k}{8000 \text{ data points}} \approx 0.246 = 24.6\%$$

24.6 Percent of training data has an income > 50k

In this case, we have 24.6 percent of the data have incomes greater than 50k, and the rest have less than 50k, it means that data set is not balanced, majority of data points will fall into the "less 50k" class, this might be cause a skewed model performance metrics.a model that archived 70% is not a good model here, because due to the imbalanced data, the majority of predicting results might be fall into the class which incomes is less than 50k, achieving high accuracy by frequently guessing this class most of time. The model will perform poorly on correctly identifying the class "incomes greater than 50k"
Each data point has 83 dimensions

▶ Q6 Norms and Distances [2pt] Distances and vector norms are closely related concepts. For instance, an L_2 norm of a vector $\mathbf{x}$ (defined below) can be intepretted as the Euclidean distance between $\mathbf{x}$ and the zero vector:

$$||x||_2 = \sqrt{\sum_{i=1}^{d} x_i^2} \tag{3}$$

Given a new vector $\mathbf{z}$, show that the Euclidean distance between $\mathbf{x}$ and $\mathbf{z}$ can be written as an L_2 norm. [kNN implementation note for later, you can compute norms efficiently with `numpy` using `np.linalg.norm`]

We can assume that vector $X = [x_1, x_2, \cdots, x_i]$ and vector $z = [z_1, z_2, \cdots z_i]$, Now let's Calculate the Euclidean distance between $\vec{x}$ and $\vec{z}$ as below:

$$ED(\vec{x}, \vec{z}) = \sqrt{(x_1 - z_1)^2 + (x_2 - z_2)^2 + \cdots, (x_i - z_i)^2}$$

$$\Rightarrow \sqrt{\sum_{i=1}^{d} (x_i - z_i)^2}$$

$$\Rightarrow \|x\|_2$$

As given L_2 norm of a vector X can be interpreted as the Euclidean distance between $\vec{x}$ and $\vec{0}$ as following

$$ED(\vec{x}, \vec{0}) = \sqrt{(x_1 - \vec{0})^2 + (x_2 - 0_2)^2 + \cdots, (x_i - 0_2)^2}$$

$$\Rightarrow \sqrt{\sum_{i=1}^{d} (x_i - 0)^2} = \sqrt{\sum_{i=1}^{d} (x_i)} = ||x_i||_2$$

Therefore, $\sqrt{\sum_{i=1}^{d} (x_i - z_i)^2}$ essentially represents the L_2 norm of the vector $(X - Z)$, which is the way to compute distances between 2 vectors is Euclidean distance. Thus the l_2 norm is the square root of the sum of squared difference, and it's equivalent to the Euclidean.

▶ Q9 Hyperparameter Search [15pt] To search for the best hyperpameters, run 4-fold cross-validation to estimate our accuracy. For each k in 1,3,5,7,9,99,999,and 8000, report:

- accuracy on the training set when using the entire training set for kNN (call this **training accuracy**),
- the mean and variance of the 4-fold cross validation accuracies (call this **validation accuracy**).

Skeleton code for this is present in `main()` and labeled as Q9 Hyperparmeter Search. Finish this code to generate these values – should likely make use of `predict`, `accuracy`, and `cross_validation`.

Questions: What is the best number of neighbors (k) you observe? When k = 1, is training error 0%? Why or why not? What trends (train and cross-valdiation accuracy rate) do you observe with increasing k? How do they relate to underfitting and overfitting?

my best number of neighbors k = 60. Theoreticaly, the trainig error should be 0%, but accordig to my report, the trainig accuray rate is not 100% but 98.66%, which is the highest training accuracy rate I got in my report, based on my analysis of the trainig data, the reason for this can be atttributed to the presence of noisy data points and outliers within the training set. For instance, in the 'Age' atribute, threre are some istances with values 0, and only 2 data points have atertribute value exceding 0.07. Another example is seen in the 'capital gain' attribute, where there is just one data point with vame of 0.93 whereas the typical range falls between 0 and 0.2. with increasig K, the trainig accuracy decreases but validation accuracy rate increases until K= 999. when 'k' is small, it essentialy memorizes the training data and tries to fit data very closely. this can be lead to Overfting, where the model perforns very well on the trainig

data but poorly on unseen data. This is why the training accuracy is high but validation accuracy is low. However, when K is large & >999, the model becomes less sensitive to individual data points and relies on a large number of neighbors to make predictions. This can lead to underfitting because the model oversimplifies the underlying patterns in the data.

```
❯ python3 knn.py
Performing 4-fold cross validation
k =     1 -- train acc = 98.66%  val acc = 80.41% (0.0012)              [exe_time = 27.00]
k =     3 -- train acc = 89.99%  val acc = 82.12% (0.0009)              [exe_time = 26.70]
k =     5 -- train acc = 88.00%  val acc = 83.20% (0.0008)              [exe_time = 26.99]
k =     7 -- train acc = 87.69%  val acc = 83.39% (0.0064)              [exe_time = 27.22]
k =     9 -- train acc = 86.89%  val acc = 83.46% (0.0027)              [exe_time = 26.49]
k =    59 -- train acc = 84.90%  val acc = 84.79% (0.0015)              [exe_time = 27.64]
k =    60 -- train acc = 84.82%  val acc = 84.88% (0.0024)              [exe_time = 27.52]
k =    61 -- train acc = 84.88%  val acc = 84.85% (0.0021)              [exe_time = 27.60]
k =    67 -- train acc = 84.86%  val acc = 84.70% (0.0023)              [exe_time = 27.65]
k =    99 -- train acc = 84.86%  val acc = 84.55% (0.0014)              [exe_time = 28.35]
k =   999 -- train acc = 82.29%  val acc = 80.36% (0.0078)              [exe_time = 44.70]
k =  8000 -- train acc = 75.41%  val acc = 75.41% (0.0031)              [exe_time = 151.97]
```

3 Debriefing (required in your report)

1. Approximately how many hours did you spend on this assignment?

   About 70 hours

2. Would you rate it as easy, moderate, or difficult?

   I would rate very difficult

3. Did you work on it mostly alone or did you discuss the problems with others?

   I did discuss some problems on the discord and my friend.

4. How deeply do you feel you understand the material it covers (0%–100%)?

   I feel like 65% understand the material it covers

5. Any other comments?

   No