CS434 Machine Learning and Data Mining – Homework 1

k-Nearest Neighbor Classification and Statistical Estimation

Solutions - DO NOT DISTRIBUTE

Overview and Objectives. In this homework, we are going to implement a kNN classifier and get some practice with the concepts in statistical estimation we went over in lecture. The assignment will help you understand how to apply the concepts from lecture to applications.

How to Do This Assignment.

- Each question that you need to respond to is clearly marked in a blue "Task Box" with its corresponding point-value listed.
- We prefer typeset solutions (LaTeX / Word) but will accept scanned written work if it is legible. If a TA can't read your work, they can't give you credit. Submit your solutions to Canvas as a zip including your report and any code the assignment asks you to develop. You do not need to include the data files.
- Programming should be done in Python and numpy. If you don't have Python installed in your machine, you can install it from https://www.python.org/downloads/. This is also the link showing how to install numpy: https://numpy.org/install/. You can also search through the internet for numpy tutorials if you haven't used it before. Google is your friend!

You are NOT allowed to...

- Use machine learning package such as `sklearn`.
- Use data analysis package such as `panda` or `seaborn`.
- Discuss low-level details or share code / solutions with other students.

Advice. Start early. Start early. Start early. There are two sections to this assignment – one involving working with math (20% of grade) and another focused more on programming (80% of the grade). Read the whole document before deciding where to start.

How to submit. Submit a zip file to Canvas. Inside, you will need to have all your working code and `hw1-report.pdf`. You will also submit test set predictions to a class Kaggle competition. This is a required to receive credit for Q10.

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.

▶ Q1 solution

From the Poisson distribution equation, we will have the likelihood function as below.

$$P(D|\lambda) = \prod_{i=1}^{N} P(x_i|\lambda) = \prod_{i=1}^{N} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!}$$

Take natural log (ln) on both side

$$\begin{aligned}
lnP(D|\lambda) &= ln \prod_{i=1}^{N} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!} \\
&= \sum_{i=1}^{N} ln \left(\frac{\lambda^{x_i} e^{-\lambda}}{x_i!} \right) \\
&= \sum_{i=1}^{N} [ln(\lambda^{x_i}) + ln(e^{-\lambda}) - ln(x_i!)] \\
&= \sum_{i=1}^{N} [x_i ln(\lambda) - \lambda - ln(x_i!)] \\
&= \sum_{i=1}^{N} x_i ln(\lambda) - \sum_{i=1}^{N} \lambda - \sum_{i=1}^{N} ln(x_i!) \\
&= ln(\lambda) \sum_{i=1}^{N} x_i - N\lambda - \sum_{i=1}^{N} ln(x_i!)
\end{aligned}$$

Take the derivative:

$$\begin{aligned}
\frac{d}{d\lambda} lnP(D|\lambda) &= \frac{d}{d\lambda} \left(ln(\lambda) \sum_{i=1}^{N} x_i - N\lambda - \sum_{i=1}^{N} ln(x_i!) \right) \\
&= \frac{1}{\lambda} \sum_{i=1}^{n} x_i - N
\end{aligned}$$

Set the derivative equal to zero and solve for λ.

$$\begin{aligned}
\frac{1}{\lambda} \sum_{i=1}^{N} x_i - N &= 0 \\
\frac{1}{\lambda} \sum_{i=1}^{N} x_i &= N \\
\sum_{i=1}^{n} x_i &= \lambda N \\
\lambda_{\text{MLE}} &= \frac{1}{N} \sum_{i=1}^{N} x_i
\end{aligned}$$

In lecture, we discussed how to use priors to encode beliefs about parameters *before any data is seen*. We showed

the Beta distribution was a conjugate prior to the Bernoulli – that is to say that the posterior of a Bernoulli likelihood and Beta prior is itself a Beta distribution (i.e. Beta $\propto$ Bernoulli * Beta) . The Poisson distribution also has a conjugate – the Gamma distribution. Gamma is a continuous distribution over the range $[0, \infty)$ with probability density function:

$$\texttt{Gamma}(\Lambda = \lambda; \alpha, \beta) = \frac{\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)} \qquad \forall \lambda \geq 0 \qquad \alpha, \beta > 0 \tag{2}$$

Note that $\Gamma(\alpha)$ is referred to as the Gamma *function* whereas the overall expression represents the Gamma *distribution*.

▶ 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 $\hat{\lambda}_{MAP}$.

▶ Q2 solution

We have:

$$\begin{aligned}
P(\lambda|D) &\propto P(D|\lambda)P(\lambda) \\
logP(\lambda|D) &\propto logP(D|\lambda) + logP(\lambda) \\
&\propto ln(\lambda)\sum_{i=1}^{N} x_i - N\lambda - \sum_{i=1}^{N} ln(x_i!) + ln\frac{\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)} \\
&\propto ln(\lambda)\sum_{i=1}^{N} x_i - N\lambda - \sum_{i=1}^{N} ln(x_i!) + ln\lambda^{\alpha-1} + ln(e^{-\beta\lambda}) \\
&\propto ln(\lambda)\sum_{i=1}^{N} x_i - N\lambda - \sum_{i=1}^{N} ln(x_i!) + (\alpha-1)ln\lambda - \beta\lambda
\end{aligned}$$

Take the derivative:

$$\frac{d}{d\lambda}\left(ln(\lambda)\sum_{i=1}^{N} x_i - N\lambda - \sum_{i=1}^{N} ln(x_i!) + (\alpha-1)ln\lambda - \beta\lambda\right) = \frac{1}{\lambda}\sum_{i=1}^{N} x_i - N + \frac{1}{\lambda}(\alpha-1) - \beta$$

Set the derivative equal to zero and solve for λ.

$$\begin{aligned}
\frac{1}{\lambda}\sum_{i=1}^{N} x_i - N + \frac{1}{\lambda}(\alpha-1) - \beta &= 0 \\
\frac{1}{\lambda}\left((\alpha-1) + \sum_{i=1}^{N} x_i\right) - N - \beta &= 0 \\
\frac{1}{\lambda}\left((\alpha-1) + \sum_{i=1}^{N} x_i\right) &= N + \beta \\
\lambda_{\text{MAP}} &= \frac{(\alpha-1) + \sum_{i=1}^{N} x_i}{N + \beta}
\end{aligned}$$

You can also see that if we modify the equation a little bit we will get:

$$\lambda_{\text{MAP}} = \left(\frac{\alpha-1}{\beta+N}\right) + \left(\frac{\sum_{i=1}^{N} x_i}{\beta+N}\right) = \left(\frac{\beta}{\beta+N}\right)\left(\frac{\alpha-1}{\beta}\right) + \left(\frac{N}{\beta+N}\right)\left(\frac{\sum_{i=1}^{N} x_i}{N}\right)$$

This looks like a weighted average of the mode of the gamma prior $(\frac{\alpha-1}{\beta})$ and the MLE maximizer $(\frac{\sum_{i=1}^{n} x_i}{n})$, with weights based on the proportion of β and n.

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 ($\propto$).]

▶ Q3 solution

From Bayes Rule, we have:

$$P(\lambda|D) \propto P(D|\lambda)P(\lambda)$$

Substitute $P(D|\lambda) = \left(\prod_{i=1}^{N} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!}\right)$ and $P(\lambda) = \frac{\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)}$.

$$\begin{aligned}
P(\lambda|D) &\propto \left(\prod_{i=1}^{N} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!}\right) \frac{\beta^\alpha \lambda^{\alpha-1} e^{-\beta\lambda}}{\Gamma(\alpha)} \\
&\propto \frac{\lambda^{\sum_{i=1}^N x_i} e^{-N\lambda}}{\prod_{i=1}^N (x_i!)} \lambda^{\alpha-1} e^{-\beta\lambda} \\
&\propto \lambda^{\sum_{i=1}^N x_i} e^{-N\lambda} \lambda^{\alpha-1} e^{-\beta\lambda} \\
&\propto \lambda^{\alpha-1+\sum_{i=1}^N x_i} e^{-N\lambda-\beta\lambda} \\
&\propto \lambda^{(\alpha+\sum_{i=1}^N x_i)-1} e^{-(N+\beta)\lambda} \\
&\propto Gamma(\alpha + \sum_{i=1}^{N} x_i, N+\beta)
\end{aligned}$$

So, $P(\lambda|D)$ is Gamma$(\alpha + \sum_{i=1}^N x_i, N+\beta)$

2 k-Nearest Neighbor (kNN) [50pts]

In this section, we'll implement our first machine learning algorithm of the course – k Nearest Neighbors. We are considering a binary classification problem where the goal is to classify whether a person has an annual income more or less than $50,000 given census information. Test results will be submitted to a class-wide Kaggle competition. As no validation split is provided, you'll need to perform cross-validation to fit good hyperparameters.

Data Analysis. We've done the data processing for you for this assignment; however, getting familiar with the data before applying any algorithm is a *very* important part of applying machine learning in practice. Quirks of the dataset could significantly impact your model's performance or how you interpret the outcome of your experiments. For instance, if I have an algorithm that achieved 70% accuracy on this task – how good or bad is that? We'll see!

We've split the data into two subsets – a **training** set with 8,000 labelled rows, and a **test** set with 2,000 unlabelled rows. These can be downloaded from the data tab of the HW1 Kaggle as "train.csv" and "test_pub.csv" respectively. Both files will come with a header row, so you can see which column belongs to which feature.

Below you will find a table listing the attributes available in the dataset. We note that categorizing some of these attributes into two or a few categories is reductive (e.g. only 14 occupations) or might reinforce a particular set of social norms (e.g. categorizing sex or race in particular ways). For this homework, we reproduced this dataset from its source without modifying these attributes; however, it is useful to consider these issues as machine learning practitioners.

attribute name: type. *list of values*
id: numerical. Unique for each point. Don't use this as a feature (it will hurt, badly).
age: numerical. ***workclass***: categorical. *Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay* ***education-num***: ordinal. *1:Preschool, 2:1st-4th, 3:5th-6th, 4:7th-8th, 5:9th, 6:10th, 7:11th, 8:12th, 9:HS-grad, 10:Some-college, 11:Assoc-voc, 12:Assoc-acdm, 13:Bachelors, 14:Masters, 15:Prof-school, 16:Doctorate* ***marital-status***: categorical. *Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse* ***occupation***: categorical. *Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces* ***relationship***: categorical. *Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried* ***race***: categorical. *White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black* ***sex***: categorical. *0:Male, 1:Female* ***capital-gain***: numerical. ***capital-loss***: numerical. ***hours-per-week***: numerical. ***native-country***: categorical. *United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinada&Tobago, Peru, Hong, Holand-Netherlands*
income: ordinal. 0: <=50K, 1: >50K This is the class label. Don't use this as a feature.

Our dataset has three types of attributes – numerical, ordinal, and nominal. Numerical attributes represent continuous numbers (e.g. hours-per-week worked). Ordinal attributes are a discrete set *with a natural ordering*, for instance different levels of education. Nominal attributes are also discrete sets of possible values; however, there is no clear ordering between them (e.g. native-country). These different attribute types require different preprocessing. As discussed in class, numerical fields have been normalized.

For nominal variables like workclass, marital-status, occupation, relationship, race, and native-country, we've transformed these into one column for each possible value with either a 0 or a 1. For example, the first instance in the training set reads: `[0, 0.136, 0.533, 0.0, 0.659, 0.397, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]` where all the zeros and ones correspond to these binarized variables. The following questions guide you through exploring the dataset and help you understand some of the steps we took when preprocessing the data.

▶ `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.

▶ Q4 solution

Categorical features represent features that can take on a discrete set of values. Representing these as:

- **Ordinal Encodings** such as mapping 1=Private, 2=State-gov, and 3=Never-worked as in the example above induce an ordering among the feature values. Consider this sort of encoding for the four example entries above:

	workclass
Example 1	Private
Example 2	State-gov
Example 3	Never-worked
Example 4	State-gov

The original field

	workclass
Example 1	1
Example 2	2
Example 3	3
Example 4	2

Fields after ordinal encoding

  Consider the Euclidean distance between these four examples. Examples 2 and 4 share the same workclass and have zero distance. Examples 1 and 2 have different workclasses (Private vs State-gov) and distance of 1. Examples 1 and 3 also have different workclasses (Private vs Never-worked), but these have a distance of 2. Did we intend for Private work to be *more* different from Never-worked than State-gov? Likely not.

- **One-hot Encodings** In contrast, distances between one-hot encodings are straightforward. If two examples have different workclasses, they have a distance of $\sqrt{2}$ and a distance of 0 otherwise. This encoding assumes all workclasses are equally distant from one another.

▶ 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]

▶ Q5 solution

We have 1967 positive labels (>50K) in the training set, which is around 24.59%. We can see that the data is biased toward the negative label (≤50K). If we consider a baseline model that just predicts negative for every example, it would achieve an approximately 75% accuracy rate. So a model performing at an accuracy rate of 70% is likely to be a poor model in this case.

The data has 85 dimensions.

2.1 k-Nearest Neighbors

In this part, you will need to implement the k-NN classifier algorithm with the Euclidean distance. The kNN algorithm is simple as you already have the preprocessed data. To make a prediction for a new data point, there are three main steps:

1. Calculate the distance of that data point to every training example.
2. Get the k nearest points (i.e. the k with the lowest distance).
3. Return the majority class of these neighbors as the prediction

Implementing this using native Python operations will be quite slow, but lucky for us there is the `numpy` package. `numpy` makes matrix operations quite efficient and easy to code. How will matrix operations help us? The next question walk you through figuring out the answer. Some useful things to check out in `numpy`: broadcasting, `np.linalg.norm` `np.argsort()` or `np.argpartition()`, and array slicing.

▶ 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`]

▶ Q6 solution

Given vectors $\mathbf{x}$ and $\mathbf{z}$, we can write the Euclidean distance between these vectors as:

$$d(\mathbf{x}, \mathbf{z}) = \sqrt{\sum_{i=1}^{d} (x_i - z_i)^2}$$

This nearly matches the L2-vector norm equation; however, there is a $x_i - z_i$ term being squared rather than simply the i'th element of some vector. Let $\mathbf{y} = \mathbf{x} - \mathbf{z}$. As vector subtraction is element-wise, we will have:

$$||\mathbf{y}||_2 = \sqrt{\sum_{i=1}^{d} y_i^2} = \sqrt{\sum_{i=1}^{d} (x_i - z_i)^2} = d(\mathbf{x}, \mathbf{z})$$

In kNN, we need to compute distance between every training example $\mathbf{x}_i$ and a new point $\mathbf{z}$ in order to make a prediction. As you showed in the previous question, computing this for one $\mathbf{x}_i$ can be done by applying an arithmetic operation between $\mathbf{x}_i$ and $\mathbf{z}$, then taking a norm. In `numpy`, arithmetic operations between matrices and vectors are sometimes defined by "broadcasting", even if standard linear algebra doesn't allow for them. For example, given a matrix X of size $n \times d$ and a vector $\mathbf{z}$ of size d, `numpy` will happily compute $Y = X - z$ such that Y is the result of the vector z being subtracted from each row of X. **Combining this with your answer from the previous question can make computing distances to every training point quite efficient and easy to code.**

▶ Q7 Implement kNN Classifier [10pts] Okay, it is time to get our hands dirty. Let's write some code! Implement k-Nearest Neighbors using Euclidean distance by completing the skeleton code provided in `knn.py`. Specifically, you'll need to finish:

- `get_nearest_neighbors(example_set, query, k)`
  Given a $n \times d$ `example_set` matrix where each row represents one example point, return the indices of the k nearest examples to the `query` point. This is where the bulk of the computation will happen in your algorithm so you are strongly encouraged to review the paragraph above this task box to get hints for how to do it efficiently in numpy.
- `knn_classify_point(examples_X, examples_y, query, k)`
  Given a $n \times d$ `example_set` matrix where each row represents one example point and a $n \times 1$ column-vector with these points' corresponding labels, return the prediction of a kNN classifier for the `query` point. Should use the previous function.

The code to load the data, predict for each point in a matrix of queries, and compute accuracy is already provided towards the end of the file. You'll want to read over these carefully.

2.2 K-Fold Cross Validation

We do not provide class labels for the test set or a validation set, so you'll need to implement K-fold Cross Validation[1] to check if your implementation is correct and to select hyperpameters. As discussed in class, K-fold Cross Validation divides the training set into K segments and then trains K models – leaving out one of the segments each time and training on the others. Then each trained model is evaluated on the left-out fold to estimate performance. Overall performance is estimated as the mean and variance of these K fold performances.

[1] Note that K here has nothing to do with k in kNN – just overloaded notation.

▶ Q8 Implement k-fold Cross Validation [8pts] Next we'll be implementing k-fold cross validation by finishing the following function in `knn.py`:

- `cross_validation(train_X, train_y, num_folds, k)`
  Given a $n \times d$ matrix of training examples and a $n \times 1$ column-vector of their corresponding labels, perform K-fold cross validation with `num_folds` folds for a k-NN classifier. To do so, split the data in `num_folds` equally sized chunks then evaluate performance for each chunk while considering the other chunks as the training set. Return the average and variance of the accuracies you observe.

For simplicity, you can assume `num_folds` evenly divides the number of training examples. You may find the numpy split and vstack functions useful.

▶ 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?

▶ Q9 solution

```
Performing 4-fold cross validation
k =    1 -- train acc = 98.74%  val acc = 78.75% (0.0031) [exe_time = 36.06]
k =    3 -- train acc = 89.01%  val acc = 80.41% (0.0019) [exe_time = 36.31]
k =    5 -- train acc = 86.90%  val acc = 81.54% (0.0043) [exe_time = 36.51]
k =    7 -- train acc = 86.22%  val acc = 81.54% (0.0050) [exe_time = 37.27]
k =    9 -- train acc = 85.49%  val acc = 81.70% (0.0036) [exe_time = 37.77]
k =   99 -- train acc = 83.25%  val acc = 82.62% (0.0032) [exe_time = 38.28]
k =  999 -- train acc = 82.30%  val acc = 81.95% (0.0055) [exe_time = 44.63]
k = 8000 -- train acc = 75.41%  val acc = 75.41% (0.0031) [exe_time = 82.69]
```

The best number of neighbors that I got is 99. When k = 1, the training error is not 0% in this case. For a training set with unique datapoints, we would expect each training point to be its own nearest neighbor; however, this dataset includes some duplicate points where all features are the same but the labels are different. While evaluating training error, these points are sometimes mispredicted.

When k increases, the training accuracy decreases but the cross-valdiation accuracy will start increasing but also being to drop after a point. With a small k, the model seems to overfit the data (high train accuracy but low validation accuracy) and when k is too large, the model will underfit the data (low train and validation accuracy).

2.3 Kaggle Submission

Great work getting here. In this section, you'll submit the predictions of your best model to the class-wide Kaggle competition. You are free to make any modification to your kNN algorithm to improve performance; however, it must remain a kNN algorithm! You can change distances, weightings, select subsets of features, etc.

▶ Q10 Kaggle Submission [10pt]. Code at the end of `main()` outputs predictions for the test set to `test_predicted.csv`. Decide on hyperparameters and submit a set of predictions to Kaggle that outperforms the baseline on the public leaderboard. Predictions are formatted as a two-column CSV as below:

```
id,income
0,0
1,0
2,0
3,0
4,0
5,1
6,0
.
.
.
```

You may submit up to 5 times a day. In your report, tell us what modifications you made to kNN for your final submission. Note that the public leaderboard shows results on only *half* the test data, whereas a separate private leaderboard shows results for the whole set.

Extra Credit and Bragging Rights [2.5pt Extra Credit]. The TA has made a submission to the leaderboard as well. Any submission outperforming the TA on the *private* leaderboard at the end of the homework period will receive 2.5 extra credit points on this assignment. Further, the top 5 ranked submissions will "win HW1" and receive bragging rights.

3 Debriefing (required in your report)

1. Approximately how many hours did you spend on this assignment?
2. Would you rate it as easy, moderate, or difficult?
3. Did you work on it mostly alone or did you discuss the problems with others?
4. How deeply do you feel you understand the material it covers (0%–100%)?
5. Any other comments?