Linear Algebra

1. The symmetric matrix A over $\mathbb{R}$ is PSD if for every vector v: $v^T A v \geq 0$.
   a. Prove that $A\ is\ PSD \leftrightarrow \exists X. A = XX^T$:
      1) $A\ is\ PSD \rightarrow \exists X. A = XX^T$
      Since A is a PSD, we can learn that any $\lambda \in eigenvalues(A)$ is nonnegative:
      $\forall \lambda. Av = \lambda v$ and multiplying by v_T from the left will result $v^t Av = \lambda v^T v = \lambda |v|^2$. Since A is PSD the expression $v^t Av \geq 0$ so therefore $\lambda \geq 0$
      Using the hint, we will rewrite A as QDQ^T with Q being orthogonal matrix whose columns are eigenvectors of A and D is a diagonal matrix with eigenvalues of A. we will also define matrix S a diagonal matrix with the square roots of eigenvalues of A:
      $$S = \begin{pmatrix} \sqrt{\lambda_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sqrt{\lambda_n} \end{pmatrix}, D = \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{pmatrix} \rightarrow D = SS = SS^T$$
      So now we can define $X = QS \rightarrow X^T = Q^T S^T$ so we can get to the point:
      $$A = QDQ^T = QBB^T Q^T = XX^T$$

      2) $\exists X. A = XX^T \rightarrow A\ is\ PSD$:
      Let A be matrix that can written as $A = XX^T$ and let $v \in \mathbb{R}^n$ so $v^T Av = v^T XX^T v$
      Since $v^T X$ and $X^T v$ are multiplication of the same elements (simply in transpose) we can mark $v^T X = u = (u_1, \cdots, u_n)$ and just as well $X^T v = u^T = (u_1, \cdots, u_n)^T$ hence we can see:
      $$v^T Av = uu^T = (u_1, \cdots, u_n)\begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix} = \sum_{i-1}^{n} u_i^2 \rightarrow v^T Av \geq 0 \rightarrow A\ is\ PSD\ by\ definition$$
   b. To show that $\forall \alpha, \beta \geq 0$ and PSD matrices A,B the matrix $\alpha A + \beta B$ is also PSD we will show that $\forall \gamma \geq 0$ and $D \in \mathbb{R}^{n \times n}$ PSD: $C = \gamma D$ is also a PSD: $v^T Cv = v^T \gamma Dv = |\gamma| v^T Dv\ so\ we\ get\ (\forall v. v^T Dv \geq 0 \rightarrow\ \forall v. v^T Cv)$.
      Now all we need to show is that $\forall C, D$ PSD matrices the sum $S = C + D$ is also PSD (when $C = \alpha A, D = \beta B$).
      A sum of symmetric matrices is also symmetric and over $\mathbb{R}$. we need to show that $\forall v. v^t Sv \geq 0$. let's mark $\tilde{c} = v^T Cv$, $\tilde{d} = v^T Dv$. We already know $\tilde{c}, \tilde{d} \geq 0$ so $v^T Sv = v^t (C + D)v = v^T Cv + v^T Dv = \tilde{c} + \tilde{d} \geq 0$.
      $$v^T(\alpha A + \beta B)v = v^T(C + D)v = v^T Sv \geq 0 \rightarrow S = C + D = \alpha A + \beta B \in \{PSD\}$$
      The definition of a vector space includes the requirement for having an inverse element for each element in space : Having an Inverse element should satisfy $(A + Inv(A) = 0$ in out case $Inv(A) = -A$.
      $v^T(Inv(A))v = v^T(-A)v = -(v^T Av)$ and since $v^T Av \geq 0$ we gent $v^T\big(Inv(A)\big)v \leq 0$
      If we can fine v that satisfies $v^T Av > 0$ we get $-v^T Av < 0$ hence $Inv(A)$ isn't a PSD. in the set of all PSD matrices we will have at least one positive definite matrix (let's mark P) which it's Inverse (-P) isn't a PSD.

Calculus and Probability

1. Let $Y = \max(X_1, \cdots X_n)$ when $X_1, \cdots, X_n$ are IID $U([0,1])$ continuous random variables.
   It is known (learned in intro to statistics for electric engineers) that give IID set of variables the maximum of them satisfies:
   $F_Y(t) = \big(F_{X_i}(t)\big)^n$ and $f_Y(t) = n \cdot \big(F_{X_i}(t)\big)^{n-1}$.
   In our case since $X_i \sim U([0,1])$: $F_{X_i}(t) = \begin{cases} 0 & t < 0 \\ t & t \in [0,1] \\ 1 & 1 < t \end{cases}$ hence $F_Y(t) = \begin{cases} 0 & t < 0 \\ t^n & t \in [0,1] \\ 1 & 1 < t \end{cases}$ and $f_Y(t) = \begin{cases} 0 & t < 0 \\ nt^{n-1} & t \in [0,1] \\ 0 & 1 < t \end{cases}$ which is β distribution with $\alpha = n, \beta = 1$:
   For β distribution we get $\mathbb{E}[Y] = \frac{\alpha}{\alpha+\beta} = \frac{n}{n+1}$ (when n increase the expected value increases too: makes sense)
   We also get $Var(Y) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+b+1)} = \frac{n}{(n+1)^2(n+2)}$ (when n increases the variance decreases makes sense)

Decision Rules and Concentration Bounds

1.

a. Let X and Y be random variables where Y can take values in $\{1, \cdots, L\}$. Let ℓ_{0-1} be the 0-1 loss function defined in class. We will show that $h = argmin_{f:X\to Y}(\mathbb{E}[\ell_{0-1}(f(X), Y)])$ is given by $h(x) = \arg\max_{i\in Y} P[Y = i|X = x]$.

For that we will understand the minimizing value of $\mathbb{E}[\ell_{0-1}(f(X), Y)]$:

$$L(h) = \mathbb{E}[\ell_{0-1}(f(X), Y)] = \sum_{x,y} P(X = x, Y = y) \cdot \ell_{0-1}(f(x), y)$$

Since we have no control over $P(X = x, Y = y)$ we will minimize the expression by choosing $f(x)$ which minimizes $\ell_{0-1}(f(x), y)$

$$L(h) = \sum_{y\in Y} P[X = \hat{x}, Y = y] \cdot \ell_{0-1}(f(x), y) = P[X = \hat{x}] \sum_{y\in Y} P[Y = y|X = \hat{x}]\ell_{0-1}(f(\hat{x}), y)$$

$$L(h) = P[x = \hat{x}]\begin{cases} 1 - P[Y = 1|X = \hat{x}] & f(\hat{x}) = 1 \\ \vdots & \vdots \\ 1 - P[Y = L|X = \hat{x}] & f(\hat{x}) = L \end{cases} = P[X = \hat{x}]\left(1 - P\left[Y = f(x)\middle|X = \hat{X}\right]\right)$$

Minimizing $L(h)$ is equivalent to maximizing $P\left[Y = f(x)\middle|X = \hat{X}\right]$ therefore:

$$h(x) = argmax_{i\in Y}(P[Y = i \mid X = x])$$

b. Let X, Y be random variables where Y can take values in $\{0,1\}$. Let Δ be an asymmetric loss function satisfying:

$\Delta(y, \hat{y}) = \begin{cases} 0 & y = \hat{y} \\ a & y = 0, \hat{y} = 1 \\ b & y = 1, \hat{y} = 0 \end{cases}$ where $a, b \in (0,1]$. We will calculate the optimal decision rule for Δ.

Using the same steps as before, just remembering now we have unbiased Δ function meaning we will have another factor in $h(x)$ *considering the unbiased loss function.*

$$L(h) = \mathbb{E}[\Delta(f(X), Y)] = \sum_{x,y} P(X = x, Y = y) \cdot \Delta(f(x), y)$$

$$L(h) = \sum_{y\in Y} P[X = \hat{x}, Y = y] \cdot \Delta(f(x), y) = P[X = \hat{x}]\left(P[Y = 0|X = \hat{x}]\Delta(f(\hat{x}), 0) + P[Y = 1|X = \hat{x}]\Delta(f(\hat{x}), 1)\right)$$

$$L(h) = P[X = \hat{x}]\begin{cases} P[Y = 1|X = \hat{x}] \cdot b & f(\hat{x}) = 0 \\ P[Y = 0|X = \hat{x}] \cdot a & f(\hat{x}) = 1 \end{cases} = P[X = \hat{x}]\begin{cases} 1 - P[Y = 0|X = \hat{x}] \cdot b & f(\hat{x}) = 0 \\ 1 - P[Y = 1|X = \hat{x}] \cdot a & f(\hat{x}) = 1 \end{cases}$$

$$P[X = \hat{x}]\left((1 - f(x))b + (f(x))a\right)\left(1 - P\left[Y = (f(x))|X = \hat{x}\right]\right)$$

$$h(x) = argmax_{y\in Y}\left(((1 - y)b + ay)P[Y = y|X = \hat{x}]\right)$$

2.

(25 pts) Let $\mathbf{X} = (X_1, \ldots, X_d)^\mathsf{T}$ be a vector of random variables. $\mathbf{X}$ is said to have a **multivariate normal (or Gaussian) distribution** with mean $\boldsymbol{\mu} \in \mathbb{R}^d$ and a $d \times d$ positive definite[1] covariance matrix Σ, if its probability density function is given by

$$f(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\mathsf{T}\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

where $\mathrm{E}[X_i] = \mu_i$ and $cov(X_i, X_j) = \Sigma_{ij}$ for all $i, j = 1, \ldots, d$. We write this as $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$.

In this question, we generalize the decision rule we have seen in the recitation to more cases. Assume that the data is $\langle \mathbf{x}, y \rangle$ pairs, where $\mathbf{x} \in \mathbb{R}^d$ and $y \in \{0, 1\}$. Denote by $f_0(\mathbf{x})$ and $f_1(\mathbf{x})$ the probability density functions of $\mathbf{x}$ given each of the label values. It is known that f_0, f_1 are multivariate Gaussian:

$$f_0(\mathbf{x}) = f(\mathbf{x}; \boldsymbol{\mu}_0, \Sigma_0),$$
$$f_1(\mathbf{x}) = f(\mathbf{x}; \boldsymbol{\mu}_1, \Sigma_1).$$

Also, the probability to sample a positive sample (i.e. $y = 1$) is p. Assume throughout the question that Σ_0, Σ_1 are positive diagonal matrices.

a. Not for submission

b. Assuming $\Sigma_0 = \Sigma_1 = \Sigma$, $P(Y=1) = p$ with decision rule $P[y=1|X] > P[y=0|X] \leftrightarrow y = 1$.
We need to find an equivalent simpler decision rule. (Orange marks are for elements that can be ignored since they are equal in both sides of inequality.
Applying Bayes rule on both sides of the equation:

$$\frac{f_{X|Y}(x|1)\cdot\mathbb{P}(Y=1)}{f_X(x)} > \frac{f_{X|Y}(x|0)\cdot\mathbb{P}(Y=0)}{f_X(x)} \leftrightarrow f_1(x)p > f_0(x)(1-p)$$

$$p\frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}e^{-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)} > (1-p)\frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}e^{-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)}$$

$$\frac{p}{1-p} > e^{-\frac{1}{2}((x-\mu_0)^T\Sigma^{-1}(x-\mu_0)+(x-\mu_1)^T\Sigma^{-1}(x-\mu_1))}$$

$$2\,ln\left(\frac{p}{1-p}\right) > (x-\mu_1)^T\Sigma^{-1}(x-\mu_1) - (x-\mu_0)^T\Sigma^{-1}(x-\mu_0)$$

$$2\,ln\left(\frac{p}{1-p}\right) > \mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0 + (\mu_0^T - \mu_1^T)\Sigma^{-1}x + x^T\Sigma^{-1}(\mu_0 - \mu_1)$$

Lest define $v = \mu_0 - \mu_1$. Since Σ is diagonal matrix, we get $v^\mathrm{T}\Sigma x = x^T\Sigma v$ and aswell for Σ^{-1} so we get:
for convenient we mark $q = \frac{p}{1-p}$, $\mu_0^T\Sigma^{-1}\mu_0 = \gamma_0, \mu_1^T\Sigma^{-1}\mu_1 = \gamma_1$

$$ln(q) + \frac{1}{2}(\gamma_0 - \gamma_1) > v^T\Sigma^{-1}x$$

When numeric values will be provided the left section will be calculated once and $v^T\Sigma^{-1}$ will be vector simply needed to be multiplied from the left with x and compared with threshold.

c. The decision boundary is d-dimensional splitting the d-dimensional space to 2 areas:
When $d = 1$ the decision boundary matches the example from class (lecture & recitation): a vertical line
When $d = 2$ the decision boundary is a line depended on both x and y.
When $d = 3$ the decision boundary is a surface 3 dependent x, y and z.

d. Assuming now $d = 1$ and $\mu = \mu_0 = \mu_1$, $\sigma_0 \neq \sigma_1$. We will find decision rule whenever $f_i(x) = f(x{:}\,\mu, \sigma_i)$
Since $P[y=1|X] > P[y=0|X] \leftrightarrow y = 1$ we get (after applying Bayes rule):

$$f(x;\mu,\sigma_1^2)\cdot p > f(x;\mu,\sigma_0^2)\cdot(1-p) \rightarrow \frac{p}{\sqrt{2\pi\cdot\sigma_1^2}}e^{-\frac{(x-\mu)^2}{2\sigma_1^2}} > \frac{1-p}{\sqrt{2\pi\cdot\sigma_0^2}}e^{-\frac{(x-\mu)^2}{2\sigma_0^2}} \rightarrow \frac{\sigma_0}{\sigma_1}\frac{p}{(1-p)} > e^{\frac{(x-\mu)^2}{2}\left(\frac{1}{\sigma_1^2}-\frac{1}{\sigma_0^2}\right)}$$

We got $\frac{2\,ln\left(\frac{\sigma_0 p}{\sigma_1(1-p)}\right)}{\left(\frac{1}{\sigma_1^2}-\frac{1}{\sigma_0^2}\right)} > (x-\mu)^2$ and we can already understand that since x is squared, we will receive two thresholds. Our decision boundary will be from the type Y=1 $\leftrightarrow$ x $\in (t_{min}, t_{max})$. for convenient we mark $q = \frac{p}{1-p}$, $\sigma = \frac{\sigma_0}{\sigma_1}$

$$t_{\min} = \mu - \sqrt{\frac{2\ln(\sigma q)}{\left(\frac{1}{\sigma_1^2}-\frac{1}{\sigma_0^2}\right)}} \qquad t_{\max} = \mu + \sqrt{\frac{2\ln(\sigma q)}{\left(\frac{1}{\sigma_1^2}-\frac{1}{\sigma_0^2}\right)}}$$

Programming Assignment

1.

Visualizing the Hoeffding bound (10 pts).

(a) Use **numpy** to generate $N \times n$ matrix of samples from $Bernoulli(1/2)$. Calculate for each row the empirical mean, $\bar{X}_i = \frac{1}{n}\sum_{j=1}^{n} X_{i,j}$, where $N = 200000$ and $n = 20$.

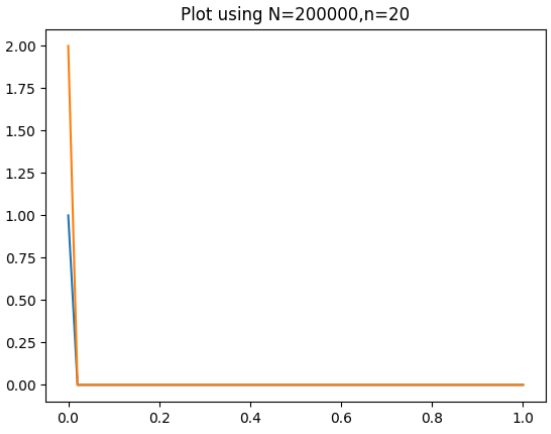
(b) Take 50 values of $\epsilon \in [0,1]$ (**numpy.linspace(0,1, 50)**), and calculate the empirical probability that $|\bar{X}_i - 1/2| > \epsilon$. Plot the empirical probability as a function of ϵ.

(c) Add to your plot the Hoeffding bound of that probability, as a function of ϵ.

Submit your plots (no need to submit code for this question).

Blue plot represents the empiric results and orange plot represents the Hoeffding bound.

```
def question_1(N: int = 200_000,
               n: int = 20,
               values_count: int = 50,
               possible_values: list[int] = [0, 1]) -> None:

    def get_averages(N: int, n: int, possible_values: list[int]) -> list[float]:
        averages = []
        for i in range(n):
            sum = 0
            for j in range(N):
                x = choice(possible_values)
                sum += x
            averages.append(sum/N)
        return averages
    print(f'starting run for Q1: with N={N},n={n}')
    print('\tCalculating avereges')
    averages = get_averages(N, n, possible_values)
    epsilons = linspace(0, 1, values_count)
    empiric_results, hoeffding = [], []

    print('\tCalculating empiric results and hoeffding bounds\n')
    for epsilon in epsilons:
        y = 0
        for x in averages:
            y = y+1 if fabs(x-0.5) > epsilon else y
        empiric_results.append(y/n)
        hoeffding.append(2*pow(e, -2*epsilon**2*N))

    plot.plot(epsilons, empiric_results)
    plot.plot(epsilons, hoeffding)
    plot.title(f'Plot using N={N},n={n}')
    plot.show()
```



2.

```
k=1: 88.3%
k=2: 88.3%
k=3: 88.2%
k=4: 88.4%
k=5: 87.1%
```

a. The code is attached below:

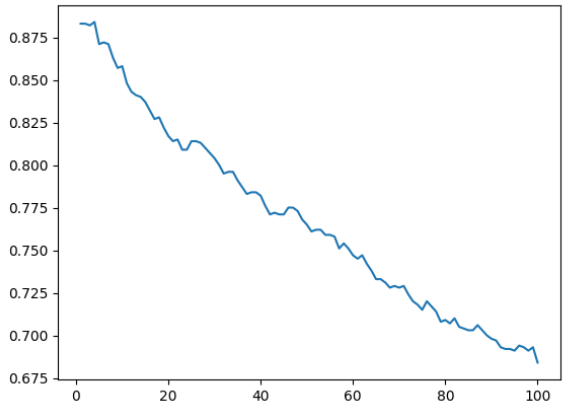
b. Using Random decision, one will choose a label out of 10 nearest neighbors hence the probability to get it right is 10%. using the algorithm (with $n = 1{,}000\ and\ k = 10$) the rate of correct predictions was 85.8% which is much better than 10% random decision.

```
n=100: 66.7%
n=200: 76.0%
n=300: 80.8%
n=400: 83.1%
n=500: 83.0%
n=600: 84.1%
n=700: 85.4%
n=800: 86.4%
n=900: 87.8%
n=1000: 88.3%
n=1100: 88.4%
n=1200: 88.7%
```

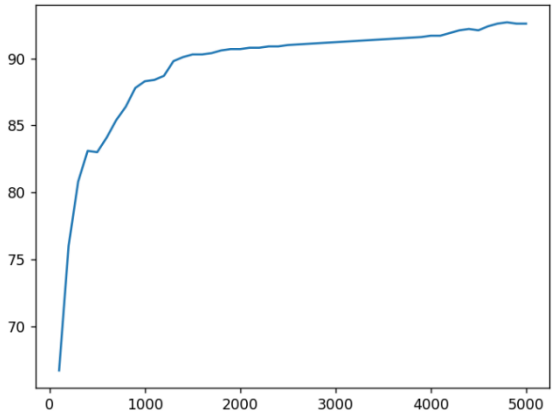
c. We can see that when using lower k values we get better prediction (getting higher prediction accuracy) with optimum of $k = 1$. Using high k value increases the possible noise (just like increasing searching radius while searching a needle in a pile of hay). It makes sense that predicting based on the closest neighbor will have better performance than having more options (and more noise). The noise can be observed in the plot.

d. We can see that increasing the number of training images (and labels) increases prediction accuracy. that does make sense, since we have a wider set of examples (for example using a data set of one image-label couple will result a single prediction). We can also observe that the improvement in accuracy is more significant when dealing with lower values of n. with the increase of n the accuracy increase (of course) but the difference of the accuracy gained by the increasing of n (or in mathematic terms $\frac{d}{dn}(accuracy)$) decreases.

```python
def question_2():

    def label_prediction(train_set, label_set, query_image, k):
        @dataclass
        class Record:
            dist: float
            label: str

        records: list[Record] = []
        for image, label in zip(train_set, label_set):
            records.append(Record(dist=linalg.norm(image - query_image),
                                  label=label))
        records.sort(key=lambda rec: rec.dist)
        labels = [r.label for r in records][:k]
        counter = {
            label: labels.count(label)
            for label in labels
        }
        return max(counter, key=counter.get)

    def single_test(n: int, k: int, train, train_labels, test, test_labels):
        current_train = train[:n]
        current_labels = train_labels[:n]
        hits, m = 0, len(test)
        for query_image, matching_label in zip(test, test_labels):
            p = label_prediction(current_train, current_labels, query_image, k)
            hits = hits+1 if p == matching_label else hits
        return hits/m

    figure, axes = plot.subplots(nrows=2, ncols=1)
    print(f'starting run for Q2:')
    mnist = fetch_openml('mnist_784', as_frame=False)
    print(f'\tfetched MNIST')
    data = mnist['data']
    labels = mnist['target']
    idx = random.RandomState(0).choice(70000, 11000)
    train = data[idx[:10000], :].astype(int)
    train_labels = labels[idx[:10000]]
    test = data[idx[10000:], :].astype(int)
    test_labels = labels[idx[10000:]]
    print(f'\tloadind data and labels from MNIST is complete')

    fixed_n = 1_000
    fixed_k = 1

    k_range = [k for k in range(1, 101)]
    n_range = [100*i for i in range(1, 51)]
    print(
        f'\trunning a tests with n={fixed_n},k in [{min(k_range)} , ... , {max(k_range)} ]')
    rates2 = [single_test(fixed_n, k, train, train_labels, test, test_labels)
              for k in k_range]
    axes[0].plot(k_range, rates2)
    print(
        f'\trunning a tests with k={fixed_k},n in [{min(n_range)} , ... , {max(n_range)} ]\n')
    rates3 = [
        single_test(n, fixed_k, train, train_labels, test, test_labels)
        for n in n_range
    ]
    axes[1].plot(n_range, rates3)
    figure.tight_layout()
    plot.show()
```



Accuracy by k (fixed n=1,000)



Accuracy by n (fixed k=1)