Homework 2 - Introduction to Machine Learning for Engineers

kipngeno koech - bkoech

February 21, 2025

1 Naive Bayes Parameters

Problem Statement

The naïve Bayes approach assumes that the feature vectors are independent given the label, that is, for any given data point $\mathbf{x}_i = [x_{i1}, x_{i2}, \cdots, x_{id}]^\top$ and its label y_i, we have

$$P(x_{i1}, x_{i2}, \cdots, x_{id}, y_i) = P(y_i) \prod_{j=1}^{d} P(x_{ij} \mid y_i)$$

Suppose that we are given the data set $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N]^\top$, where $\mathbf{X} \in \mathbb{R}^{N \times 4}$, consisting of N data points and 4 features, and its label $\mathbf{y} = [y_1, y_2, \cdots, y_N]^\top$. Assume that the first feature has 2, the second has 3, the third has 4, and the fourth has 5 possible values: Let $X_1 \in \{1, 2\}$, $X_2 \in \{1, 2, 3\}$, $X_3 \in \{1, 2, 3, 4\}$, and $X_4 \in \{1, 2, 3, 4, 5\}$. Also, there are 4 possible labels, i.e., $Y \in \{1, 2, 3, 4\} \forall i$.

This is what I have:

$$\text{Feature 1: } X_1 \in \{1, 2\}$$

$$\text{Feature 2: } X_2 \in \{1, 2, 3\}$$

$$\text{Feature 3: } X_3 \in \{1, 2, 3, 4\}$$

$$\text{Feature 4: } X_4 \in \{1, 2, 3, 4, 5\}$$

$$\text{Label: } Y \in \{1, 2, 3, 4\}$$

Naive's bayes classifier follow Bayesian rule:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

Every data point has 4 features and 1 label:

$$x_i = [x_{i1}, x_{i2}, x_{i3}, x_{i4}] \quad \text{and} \quad y_i$$

a. [5 points] Determine the number of free parameters θ and π that one has to estimate using the naïve Bayes framework. (We use θ for the probabilities of a feature given a label (e.g., $\theta_{3,2,4} = P(X_3 = 2 \mid Y = 4)$) and π for the class priors (e.g., $\pi_3 = P(Y = 3)$) similar to the lecture slides.)
Free parameters means those that we need to estimate.
class priors:

$$\pi_1 = P(Y = 1), \quad \pi_2 = P(Y = 2), \quad \pi_3 = P(Y = 3), \quad \pi_4 = P(Y = 4)$$

We need to estimate only three of these probabilities, since probability always sum up to one. So for class priors we have:

3 parameters to estimate

We need to estimate how likely each feature is given a label
Feature 1: it has 2 possible values, so we need to estimate 2 probabilities for each label

$$\theta_{1,1} = P(X_1 = 1 \mid Y = 1), \quad \theta_{1,2} = P(X_1 = 2 \mid Y = 1)$$

probability should sum up to one, so we only need to estimate one of the probabilities. So for feature 1 given 4 labels, we have:

4 parameters to estimate

Feature 2: it has 3 possible values, so we need to estimate 3 probabilities for each label

$$\theta_{2,1} = P(X_2 = 1 \mid Y = 1), \quad \theta_{2,2} = P(X_2 = 2 \mid Y = 1), \quad \theta_{2,3} = P(X_2 = 3 \mid Y = 1)$$

probability should sum up to one, so we only need to estimate two of the probabilities. So for feature 2 given 4 labels, we have:

$4 \times 2 =$ **8** parameters to estimate

Feature 3: it has 4 possible values, so we need to estimate 4 probabilities for each label:

$$\theta_{3,1} = P(X_3 = 1 \mid Y = 1), \quad \theta_{3,2} = P(X_3 = 2 \mid Y = 1), \quad \theta_{3,3} = P(X_3 = 3 \mid Y = 1), \quad \theta_{3,4} = P(X_3 = 4 \mid Y = 1)$$

probability should sum up to one, so we only need to estimate three of the probabilities. So for feature 3 given 4 labels, we have:

$4 \times 3 =$ **12** parameters to estimate

Feature 4: it has 5 possible values, so we need to estimate 5 probabilities for each label:

$\theta_{4,1} = P(X_4 = 1 \mid Y = 1), \quad \theta_{4,2} = P(X_4 = 2 \mid Y = 1), \quad \theta_{4,3} = P(X_4 = 3 \mid Y = 1), \quad \theta_{4,4} = P(X_4 = 4 \mid Y = 1), \quad \theta_{4,5} = P(X_4 = 5 \mid Y = 1)$

probability should sum up to one, so we only need to estimate four of the probabilities. So for feature 4 given 4 labels, we have:

$$4 \times 4 = \mathbf{16} \text{ parameters to estimate}$$

So the total number of parameters to estimate is:

$$3 + 4 + 8 + 12 + 16 = \mathbf{43} \text{ parameters to estimate}$$

b. [4 points] If the features are not independent conditioned on the label, one has to estimate the entire joint distribution $P(X_1, X_2, X_3, X_4 \mid Y = i)$, for all $i \in \{1, 2, 3, 4\}$. Determine the number of free parameters that one has to estimate in such a scenario.
for each label, we have 4 features, so we need to estimate the entire joint distribution for each label. For each label, we have:
For each class $Y = i$:

$$\text{Feature 1: } X_1 \in \{1, 2\}$$

$$\text{Feature 2: } X_2 \in \{1, 2, 3\}$$

$$\text{Feature 3: } X_3 \in \{1, 2, 3, 4\}$$

$$\text{Feature 4: } X_4 \in \{1, 2, 3, 4, 5\}$$

So the total number of possible combinations for each label is:

$$2 \times 3 \times 4 \times 5 = 120$$

When we add the joint distribution for every label, it should sum up to one. So the total number of parameters to estimate is:

$$120 - 1 = \mathbf{119} \text{ parameters to estimate}$$

We have 4 labels, so the total number of parameters to estimate is:

$$4 \times 119 = \mathbf{476} \text{ parameters to estimate}$$

c. [1 point] Based on the numbers of parameters you found in parts (a) and (b), explain one advantage of assuming conditional independence.
The advantage of assuming conditional independence is that it reduces the number of parameters that we need to estimate. This makes the model less complex and easier to train. It also reduces the risk of overfitting the model.

2 Naive Bayes in Practice

In this problem we will use the naïve Bayes algorithm to classify movie reviews as positive, neutral or negative. A simple approach involves maintaining a vocabulary of words that commonly occur in movie reviews and using the frequency of their occurrence in the three classes to classify movie reviews.

We are given the vocabulary $V = \{1 : \text{“incredible”}, 2 : \text{“plot”}, 3 : \text{“great”}, 4 : \text{“amazing”}, 5 : \text{“okay”}, 6 : \text{“decent”}, 7 : \text{“movie”}, 8 : \text{“no”}, 9 : \text{“acting”}, 10 : \text{“waste”}\}$. We will use V_i to represent the ith word in V. The training set provided includes four positive reviews:

- “great movie amazing”
- “incredible movie”
- “great acting amazing plot”
- “amazing acting amazing plot”

two neutral reviews:

- “okay movie”
- “decent no amazing acting”

and two negative reviews:

- “amazing waste”
- “no movie plot”

Recall that the naïve Bayes classifier is a generative classifier, where the probability of an input $\mathbf{x} = [x_1, x_2, \cdots, x_n]^\top$ depends on its class y. In our case the input vector $\mathbf{x}$ corresponding to each movie review has $n = 10$ equal to the number of words in the vocabulary V, where each entry x_i is equal to the number of times word V_i occurs in $\mathbf{x}$.

Understanding the problem at hand:

We have three classes:

1. Positive reviews: “great movie amazing”, “incredible movie”, “great acting amazing plot”, “amazing acting amazing plot”
2. Neutral reviews: “okay movie”, “decent no amazing acting”
3. Negative reviews: “amazing waste”, “no movie plot”

We have a vocabulary of 10 words:

1. “incredible”
2. “plot”
3. “great”
4. “amazing”
5. “okay”
6. “decent”
7. “movie”
8. “no”
9. “acting”
10. “waste”

Encoding the reviews - We encode the reviews as feature vectors, where each entry in the vector corresponds to the number of times a word in the vocabulary appears in the review:

Positive reviews:

$$\text{“great movie amazing”} \rightarrow [0, 0, 1, 1, 0, 0, 1, 0, 0, 0]$$

$$\text{“incredible movie”} \rightarrow [1, 0, 0, 0, 0, 0, 1, 0, 0, 0]$$

$$\text{“great acting amazing plot”} \rightarrow [0, 1, 1, 1, 0, 0, 0, 0, 1, 0]$$

$$\text{“amazing acting amazing plot”} \rightarrow [0, 1, 0, 2, 0, 0, 0, 0, 1, 0]$$

Neutral reviews:

$$\text{“okay movie”} \rightarrow [0, 0, 0, 0, 1, 0, 1, 0, 0, 0]$$

$$\text{“decent no amazing acting”} \rightarrow [0, 0, 0, 1, 0, 1, 0, 1, 1, 0]$$

Negative reviews:

$$\text{“amazing waste”} \rightarrow [0, 0, 0, 1, 0, 0, 0, 0, 0, 1]$$

$$\text{“no movie plot”} \rightarrow [0, 1, 0, 0, 0, 0, 1, 1, 0, 0]$$

a. [2 points] Calculate the naïve Bayes estimates of $\Pr(y=1)$, $\Pr(y=2)$ and $\Pr(y=3)$ from the training data, where $y=1$ corresponds to positive reviews, $y=2$ to neutral reviews and $y=3$ to negative reviews.
We have 4 positive reviews, 2 neutral reviews and 2 negative reviews. So the prior probabilities are:

$$\Pr(y) = \frac{\text{number of reviews of class } y}{\text{total number of reviews}}$$

$$\Pr(y=1) = \frac{4}{8} = \frac{1}{2}$$

$$\Pr(y=2) = \frac{2}{8} = \frac{1}{4}$$

$$\Pr(y=3) = \frac{2}{8} = \frac{1}{4}$$

b. [1 point] List the feature vector $\mathbf{x}$ for each positive review in the training set.

$$\text{“great movie amazing”} \rightarrow [0,0,1,1,0,0,1,0,0,0]$$

$$\text{“incredible movie”} \rightarrow [1,0,0,0,0,0,1,0,0,0]$$

$$\text{“great acting amazing plot”} \rightarrow [0,1,1,1,0,0,0,0,1,0]$$

$$\text{“amazing acting amazing plot”} \rightarrow [0,1,0,2,0,0,0,0,1,0]$$

c. [2 points] In the naïve Bayes model, the likelihood of a sentence with feature vector $\mathbf{x}$ given a class c is

$$\Pr(\mathbf{x} \mid y=c) = \prod_{k=1}^{n} (\theta_{c,k})^{x_k}$$

where $\theta_{c,k}$ is the weight of word k among all words of class c. Calculate the maximum likelihood estimate of $\theta_{1,4}$, $\theta_{1,7}$, $\theta_{2,4}$, $\theta_{2,7}$, $\theta_{3,4}$ and $\theta_{3,7}$.
For each class, we need to estimate the probability of each word given the class. We can calculate the maximum likelihood estimate of $\theta_{c,k}$ as:

$$\theta_{c,k} = \frac{\text{number of times word } k \text{ appears in class } c}{\text{total number of words in class } c}$$

For class 1 (positive reviews):

$$\theta_{1,4} = \theta_{\text{positive,amazing}} = \frac{4}{13} = \frac{4}{13}$$

$$\theta_{1,7} = \theta_{\text{positive,movie}} = \frac{2}{13} = \frac{2}{13}$$

For class 2 (neutral reviews):

$$\theta_{2,4} = \theta_{\text{neutral,amazing}} = \frac{1}{6} = \frac{1}{6}$$

$$\theta_{2,7} = \theta_{\text{neutral,movie}} = \frac{1}{6} = \frac{1}{6}$$

For class 3 (negative reviews):

$$\theta_{3,4} = \theta_{\text{negative,amazing}} = \frac{1}{5} = \frac{1}{5}$$

$$\theta_{3,7} = \theta_{\text{negative,movie}} = \frac{1}{5} = \frac{1}{5}$$

d. [3 points] Given a new review “amazing movie”, decide whether it is positive, neutral or negative, based on the naïve Bayes classifier, learned from the above data.

$$\Pr(y=c \mid \mathbf{x}) \propto \Pr(\mathbf{x} \mid y=c)\Pr(y=c)$$

For the new review “amazing movie”, the feature vector is:

$$\mathbf{x} = [0,0,0,1,0,0,1,0,0,0]$$

We can calculate the probability of the review being positive, neutral or negative as:

$$\Pr(\text{class} \mid \text{word}) = \Pr(\text{word} \mid \text{class})\Pr(\text{class})$$

We can calculate the probabilities as:
class positive:

$$\Pr(\text{positive} \mid \text{word}) = \theta_{1,4}^{1} \times \theta_{1,7}^{1} \times \Pr(\text{positive})$$

$$\Pr(\text{positive} \mid \text{word}) = \left(\frac{4}{13}\right) \times \left(\frac{2}{13}\right) \times \frac{1}{2} = \frac{8}{338} = 0.0237$$

class neutral:

$$\Pr(\text{neutral} \mid \text{word}) = \theta_{2,4}^{1} \times \theta_{2,7}^{1} \times \Pr(\text{neutral})$$

$$\Pr(\text{neutral} \mid \text{word}) = \left(\frac{1}{6}\right) \times \left(\frac{1}{6}\right) \times \frac{1}{4} = \frac{1}{144} = 0.0069$$

class negative:

$$\Pr(\text{negative} \mid \text{word}) = \theta_{3,4}^{1} \times \theta_{3,7}^{1} \times \Pr(\text{negative})$$

$$\Pr(\text{negative} \mid \text{word}) = \left(\frac{1}{5}\right) \times \left(\frac{1}{5}\right) \times \frac{1}{4} = \frac{1}{100} = 0.01$$

So the review “amazing movie” is **positive**.

e. [4 points] Use Laplacian smoothing with $\alpha = 1$ to decide whether the review "decent movie" is positive, neutral or negative. In one sentence, describe the problem we would encounter if we had not used Laplacian smoothing.
Laplacian smoothing is used to avoid zero probabilities. We add a small value to the numerator and denominator to avoid zero probabilities. The probability of a word given a class is calculated as:

$$\theta_{c,k} = \frac{\text{number of times word } k \text{ appears in class } c + \alpha}{\text{total number of words in class } c + \alpha \times \text{number of words in the vocabulary}}$$

For the new review "decent movie", the feature vector is:

$$\mathbf{x} = [0, 0, 0, 0, 0, 1, 1, 0, 0, 0]$$

We can calculate the probability of the review being positive, neutral or negative as:
class positive:

$$\Pr(\text{positive} \mid \text{word}) = \theta^1_{1,6} \times \theta^1_{1,7} \times \Pr(\text{positive})$$

$$\Pr(\text{positive} \mid \text{word}) = \left(\frac{1}{23}\right) \times \left(\frac{3}{23}\right) \times \frac{1}{2} = \frac{3}{1058}$$

class neutral:

$$\Pr(\text{neutral} \mid \text{word}) = \theta^1_{2,6} \times \theta^1_{2,7} \times \Pr(\text{neutral})$$

$$\Pr(\text{neutral} \mid \text{word}) = \left(\frac{2}{16}\right) \times \left(\frac{2}{16}\right) \times \frac{1}{4} = \frac{1}{256}$$

class negative:

$$\Pr(\text{negative} \mid \text{word}) = \theta^1_{3,6} \times \theta^1_{3,7} \times \Pr(\text{negative})$$

$$\Pr(\text{negative} \mid \text{word}) = \left(\frac{1}{15}\right) \times \left(\frac{2}{15}\right) \times \frac{1}{4} = \frac{2}{900}$$

So the review "decent movie" is **neutral**.

If we had not used Laplacian smoothing, we would have encountered zero probabilities, which would have made it impossible to calculate the probabilities.

3 Logistic Regression in Practice

Given a training set $D = \left\{\left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(N)}, y^{(N)}\right)\right\}$ where $\left(\mathbf{x}^{(i)} \in \mathbb{R}^d, y^{(i)} \in \{0, 1\}\right)$ is the feature vector and the binary label for data point i, we want to find the parameters $\hat{\mathbf{w}}$ that maximize the likelihood for the training set, assuming a parametric model of the form

$$p(y = 1 \mid \mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}.$$

The conditional log likelihood of the training set is

$$L(\mathbf{w}) = \sum_{i=1}^{N} \left[y^{(i)} \log p\left(y^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{w}\right) + \left(1 - y^{(i)}\right) \log\left(1 - p\left(y^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{w}\right)\right)\right],$$

and the gradient is

$$\nabla L(\mathbf{w}) = \sum_{i=1}^{N} \left(y^{(i)} - p\left(y^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{w}\right)\right) \mathbf{x}^{(i)}. \quad (2)$$

a. [2 points] Is it possible to get a closed form for the parameters $\hat{\mathbf{w}}$ that maximize the conditional log likelihood? If it is possible, give the closed-form solution. If not, how would you compute $\hat{\mathbf{w}}$ in practice? Explain your method to find $\hat{\mathbf{w}}$ in a few sentences or provide a short pseudo-code.

It is not possible to get a closed form for the parameters $\hat{\mathbf{w}}$ that maximize the conditional log likelihood. We can use gradient descent to find the parameters $\hat{\mathbf{w}}$. The gradient of the conditional log likelihood is given by Eq. (2). We can update the parameters $\mathbf{w}$ using the update rule:

$$\mathbf{w}_{(k+1)} = \mathbf{w}_{(k)} + \alpha \nabla L(\mathbf{w}_{(k)}),$$

The pseudo-code to find the parameters $\hat{\mathbf{w}}$ is as follows:

```
Initialize w
while not converged:
    for i in range(N):
        w = w + alpha * gradL(w)
```

where α is the learning rate, N is the number of data points, and gradL($\mathbf{w}$) is the gradient of the conditional log likelihood.

For a binary logistic regression model, we predict $y = 1$ when $p(y = 1 \mid \mathbf{x}) \geq 0.5$. Assume that the decision boundary occurs when $P(y = 1 \mid \mathbf{x}, \mathbf{w}) = P(y = 0 \mid \mathbf{x}, \mathbf{w})$.

b. [4 points] Find the decision boundary, which is the set of $\mathbf{x}$ satisfying $P(y = 1 \mid \mathbf{x}, \mathbf{w}) = P(y = 0 \mid \mathbf{x}, \mathbf{w})$.

we have:

$$P(y = 1 \mid \mathbf{x}, \mathbf{w}) = P(y = 0 \mid \mathbf{x}, \mathbf{w})$$
$$\frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})} = 1 - \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$
$$\frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})} = \frac{1 + \exp(-\mathbf{w}^T \mathbf{x}) - 1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$
$$\frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})} = \frac{\exp(-\mathbf{w}^T \mathbf{x})}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$
$$1 = \exp(-\mathbf{w}^T \mathbf{x})$$
$$\log(1) = \log(\exp(-\mathbf{w}^T \mathbf{x}))$$
$$0 = -\mathbf{w}^T \mathbf{x}$$
$$\mathbf{w}^T \mathbf{x} = 0$$

The decision boundary is given by the equation $\mathbf{w}^T \mathbf{x} = 0$.

c. [1 point] Is this model a linear classifier?

yes, this model is a linear classifier. The decision boundary is a linear function of the input features $\mathbf{x}$ and the model predicts the class based on whether the input lies above or below this linear boundary.

d. [2 points] Now, let us assume that in our application of this model, our tolerance of mis-classifying samples with the true label $y = 0$ is much lower than our tolerance to mis-classifying samples with true label $y = 1$ (i.e., we do not want to estimate a data point as belonging to class 0 if its true class is 1. However, we can tolerate the error of estimating a data point belong to class 1 if its true class is 0.) For example, in predicting the presence of a disease, we may be much more tolerant of mis-classifying a healthy person (true label $y = 0$) as having the disease ($y = 1$) compared to mis-classifying a person with the disease (true label $y = 1$) as not having it ($y = 0$). Now suppose you have trained a logistic regression model to obtain the model weights $\mathbf{w}$. How would you adjust the prediction with weights $\mathbf{w}$ to decrease the mis-classification of samples with true label $y = 1$?

We can adjust the prediction threshold to decrease the mis-classification of samples with true label $y = 1$. We can increase the threshold to predict $y = 1$ only when the probability is higher than the threshold. This will decrease the number of false positives. We can adjust the threshold to increase the number of true positives and decrease the number of false positives.

4 Solving Logistic Regression

The cross-entropy loss function for a logistic regression task on a dataset $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ can be written as

$$L(\mathbf{w}) = -\sum_{i=1}^{n} \left[y_i \log p(y_i \mid \mathbf{x}_i, \mathbf{w}) + (1 - y_i) \log(1 - p(y_i \mid \mathbf{x}_i, \mathbf{w})) \right],$$

In this question, we will show that finding the MLE (maximum likelihood estimator) is equivalent to minimizing the cross-entropy loss in Eq. (3). However, unlike linear regression, it is difficult to derive a closed-form solution for logistic regression.

a. [3*points*] Starting from the definition of negative log-likelihood (Eq. (4) below), show that it is equal to the cross-entropy loss (Eq. (3)).

$$-\ell(\mathbf{w}) = -\log\left(\prod_{i=1}^{n} \left(\frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right)^{y_i} \left(1 - \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right)^{1-y_i}\right). \quad (4)$$

To show that Eq. (4) is equivalent to Eq. (3), We need to expand the terms in Eq. (4) and simplifying the expression:

$$-\ell(\mathbf{w}) = -\log\left(\prod_{i=1}^{n} \left(\frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right)^{y_i} \left(1 - \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right)^{1-y_i}\right)$$

$$-\ell(\mathbf{w}) = -\sum_{i=1}^{n} \log\left(\left(\frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right)^{y_i} \left(1 - \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right)^{1-y_i}\right)$$

$$-\ell(\mathbf{w}) = -\sum_{i=1}^{n} \log\left(\left(\frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right)^{y_i} \left(1 - \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right)^{1-y_i}\right)$$

$$-\ell(\mathbf{w}) = -\sum_{i=1}^{n} \log\left(\frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right)^{y_i} \log\left(1 - \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right)^{1-y_i}$$

$$-\ell(\mathbf{w}) = -\sum_{i=1}^{n} y_i \log\left(\frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right) + (1-y_i) \log\left(1 - \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right)$$

$$-\ell(\mathbf{w}) = -\sum_{i=1}^{n} y_i \log\left(\frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right) + (1-y_i) \log\left(1 - \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right)$$

$$-\ell(\mathbf{w}) = -\sum_{i=1}^{n} y_i \log\left(\frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right) + (1-y_i) \log\left(1 - \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}\right)$$

let:

$$p(y_i \mid \mathbf{x}_i, \mathbf{w}) = \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}$$

so we have:

$$-\ell(\mathbf{w}) = -\sum_{i=1}^{n} y_i \log p(y_i \mid \mathbf{x}_i, \mathbf{w}) + (1-y_i) \log(1 - p(y_i \mid \mathbf{x}_i, \mathbf{w}))$$

$$-\ell(\mathbf{w}) = L(\mathbf{w})$$

b. [4 points] Since it is difficult to directly optimize Eq. (3), gradient descent algorithms are usually used to find the optimum, as is discussed in the lecture. Show that the negative log-likelihood is a convex function. You may use the facts that the sum of convex functions is also convex, and that if f and g are both convex, twice differentiable and g is non-decreasing, then $g(f)$ is convex.

Hint: A function f is convex if for any $x_1, x_2 \in \text{Domain}(f)$,

$$f(tx_1 + (1-t)x_2) \le tf(x_1) + (1-t)f(x_2), \quad \forall t \in [0,1] \quad (5)$$

Also, a twice-differentiable function is convex if and only if the Hessian is positive semi-definite.

$$\text{Hessian} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

We can show that the negative log-likelihood is a convex function by showing that the Hessian is positive semi-definite. The Hessian of the negative log-likelihood is given by:
The equation we want to find derivative of is:

$$-\ell(\mathbf{w}) = -\sum_{i=1}^{n} y_i \log p(y_i \mid \mathbf{x}_i, \mathbf{w}) + (1-y_i) \log(1 - p(y_i \mid \mathbf{x}_i, \mathbf{w}))$$

Let $P(y_i \mid \mathbf{x}_i, \mathbf{w}) = p$, then the negative log-likelihood is:

$$-\ell(\mathbf{w}) = -\sum_{i=1}^{n} y_i \log p + (1-y_i) \log(1-p)$$

The gradient of the negative log-likelihood is:

$$\nabla \ell(\mathbf{w}) = \sum_{i=1}^{n} (p - y_i)\, \mathbf{x}_i$$

This is because the derivative of $\log p$ is $\frac{1}{p}$ and the derivative of $\log(1-p)$ is $\frac{-1}{1-p}$.
The Hessian of the negative log-likelihood is:

$$\nabla^2 \ell(\mathbf{w}) = \sum_{i=1}^{n} p(1-p)\mathbf{x}_i \mathbf{x}_i^T$$

The Hessian is positive semi-definite because $p(1-p) \geq 0$ and $\mathbf{x}_i \mathbf{x}_i^T$ is positive semi-definite. Therefore, the negative log-likelihood is a convex function.

We now turn our attention to another algorithm known as iterative weighted least squares, which is based on the Newton-Raphson algorithm. Let $\mathbf{w}^{(k)}$ denote the parameter vector $\mathbf{w}$ at the kth iteration. Then, the update rule of iterative weighted least-squares is as follows:

$$\mathbf{w}_{(k+1)} = \mathbf{w}_{(k)} - \left(\nabla^2 \mathcal{L}\mathbf{w}_{(k)}\right)^{-1} \nabla L(\mathbf{w}_{(k)}),$$

where $\nabla L(\mathbf{w})$ and $\nabla^2 L(\mathbf{w})$ are the gradient and Hessian of the negative log-likelihood in Eq. (3), respectively. Consider the following notations:

- $\mathbf{y}$ is an $N \times 1$ column vector with the i-th element being the label y_i,
- $\mathbf{X}$ is an $N \times d$ matrix with $\mathbf{x}_i^\top$ being the i-th row,
- $\mathbf{W}_k$ is an $N \times N$ diagonal matrix with the i-th diagonal element being $\frac{\exp(-\mathbf{w}_k^\top \mathbf{x}_i)}{(1+\exp(-\mathbf{w}_k^\top \mathbf{x}_i))^2}$,
- $\mathbf{p}_k$ is an $N \times 1$ column vector with the i-th element being $\frac{1}{1+\exp(-\mathbf{w}_k^\top \mathbf{x}_i)}$,
- $\mathbf{z}_k = \mathbf{X}\mathbf{w}_k + \mathbf{W}_k^{-1}(\mathbf{y} - \mathbf{p}_k)$.

It can be shown that:

- $\nabla L(\mathbf{w}_k) = -\mathbf{X}^\top(\mathbf{y} - \mathbf{p}_k)$,
- $\nabla^2 L(\mathbf{w}_k) = \mathbf{X}^\top \mathbf{W}_k \mathbf{X}$.

c [5 points] Using the above, prove that

$$\mathbf{w}_{k+1} = (\mathbf{X}^\top \mathbf{W}_k \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{W}_k \mathbf{z}_k.$$

Note: The name of the algorithm comes from the observation that the algorithm at each step is solving the weighted least squares problem,

$$\mathbf{w}_{k+1} = \arg\min_{\mathbf{w}_k} (\mathbf{z}_k - \mathbf{X}\mathbf{w}_k)^\top \mathbf{W}_k (\mathbf{z}_k - \mathbf{X}\mathbf{w}_k).$$

To prove the equation, we need to show that:

$$\mathbf{w}_{k+1} = (\mathbf{X}^\top \mathbf{W}_k \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{W}_k \mathbf{z}_k$$

We have:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \left(\nabla^2 \mathcal{L}\mathbf{w}_k\right)^{-1} \nabla L(\mathbf{w}_k)$$

We know that:

$$\nabla L(\mathbf{w}_k) = -\mathbf{X}^\top(\mathbf{y} - \mathbf{p}_k)$$
$$\nabla^2 L(\mathbf{w}_k) = \mathbf{X}^\top \mathbf{W}_k \mathbf{X}$$

Substituting the values of $\nabla L(\mathbf{w}_k)$ and $\nabla^2 L(\mathbf{w}_k)$ into the update rule, we get:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \left(\mathbf{X}^\top \mathbf{W}_k \mathbf{X}\right)^{-1} \left(-\mathbf{X}^\top(\mathbf{y} - \mathbf{p}_k)\right)$$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \left(\mathbf{X}^\top \mathbf{W}_k \mathbf{X}\right)^{-1} \mathbf{X}^\top(\mathbf{y} - \mathbf{p}_k)$$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \left(\mathbf{X}^\top \mathbf{W}_k \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{W}_k \mathbf{X}\mathbf{w}_k - \left(\mathbf{X}^\top \mathbf{W}_k \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{W}_k \mathbf{y}$$

$$\mathbf{w}_{k+1} = \left(\mathbf{X}^\top \mathbf{W}_k \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{W}_k \mathbf{X}\mathbf{w}_k + \left(\mathbf{X}^\top \mathbf{W}_k \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{W}_k \mathbf{X}\mathbf{w}_k - \left(\mathbf{X}^\top \mathbf{W}_k \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{W}_k \mathbf{y}$$

$$\mathbf{w}_{k+1} = \left(\mathbf{X}^\top \mathbf{W}_k \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{W}_k \mathbf{y}$$

$$\mathbf{w}_{k+1} = \left(\mathbf{X}^\top \mathbf{W}_k \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{W}_k \mathbf{z}_k$$

Therefore:

$$\mathbf{w}_{k+1} = (\mathbf{X}^\top \mathbf{W}_k \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{W}_k \mathbf{z}_k = \nabla^2 L(\mathbf{w}_k) = \mathbf{X}^\top \mathbf{W}_k \mathbf{X}$$

5 SVMs: Hinge loss and mistake bounds

Suppose we build a predictive model $\hat{y} = \text{sgn}(\mathbf{w}^\top \mathbf{x})$, where $\text{sgn}(z) = 1$ if z is positive and -1 otherwise. Here we are dealing with binary predictions where each label is in $\{-1, 1\}$. The classification loss counts the number of mistakes (i.e., points where $y \neq \text{sgn}(\mathbf{w}^\top \mathbf{x})$) that we make with $\mathbf{w}$ on a dataset. When we train the SVM model on N data points $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$, the SVM loss function can be viewed as a relaxation to the classification loss.

The hinge loss on a data point $(\mathbf{x}, y)$ is defined as:

$$\ell((\mathbf{x}, y), \mathbf{w}) = \max(0, 1 - y\mathbf{w}^\top \mathbf{x}),$$

where $\mathbf{x} \in \mathbb{R}^D$ and $y \in \{-1, 1\}$. The SVM attempts to minimize:

$$L(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \ell((\mathbf{x}_i, y_i), \mathbf{w}) + \lambda \|\mathbf{w}\|_2^2,$$

where $\lambda > 0$ is the regularization parameter.

a. [4 points] Suppose that for some $\mathbf{w}$ we have a correct prediction of y_i with $\mathbf{x}_i$, i.e., $y_i = \text{sgn}(\mathbf{w}^\top \mathbf{x}_i)$. What range of values can the hinge loss, $\ell((\mathbf{x}, y), \mathbf{w})$, take on this correctly classified example? Points that are classified correctly and which have non-zero hinge loss are referred to as margin mistakes. Also, what is the possible range of the hinge loss for an incorrectly classified example?

$$\ell((\mathbf{x}, y), \mathbf{w}) = \max(0, 1 - y\mathbf{w}^\top \mathbf{x})$$

If $y_i = \text{sgn}(\mathbf{w}^\top \mathbf{x}_i)$, then $y_i \mathbf{w}^\top \mathbf{x}_i$ is positive. Therefore, the hinge loss is:

$$\ell((\mathbf{x}, y), \mathbf{w}) = \max(0, 1 - y\mathbf{w}^\top \mathbf{x}) = \max(0, 1 - y_i \mathbf{w}^\top \mathbf{x}_i) = \max(0, 1 - \mathbf{w}^\top \mathbf{x}_i)$$

The hinge loss for a correctly classified example can take values in the range $[0, 1]$. If the example is incorrectly classified, then $y_i \neq \text{sgn}(\mathbf{w}^\top \mathbf{x}_i)$, and the hinge loss is:

$$\ell((\mathbf{x}, y), \mathbf{w}) = \max(0, 1 - y\mathbf{w}^\top \mathbf{x}) = \max(0, 1 - y_i \mathbf{w}^\top \mathbf{x}_i) = \max(0, 1 + \mathbf{w}^\top \mathbf{x}_i)$$

The hinge loss for an incorrectly classified example can take values in the range $[1, \infty)$.

b. [5 points] Let $M(\mathbf{w})$ be the number of mistakes made when we use the weight vector $\mathbf{w}$ to classify our dataset (i.e., the number of training data points for which $y_i \neq \text{sgn}(\mathbf{w}^\top \mathbf{x}_i)$). Note that $\mathbf{w}$ can be an arbitrary weight vector (not necessarily the one that solves the SVM optimization problem). Show that:

$$\frac{1}{N} M(\mathbf{w}) \leq \frac{1}{N} \sum_{i=1}^{N} \max\left(0, 1 - y_i \mathbf{w}^\top \mathbf{x}_i\right). \quad (11)$$

In other words, the average hinge loss on our dataset is an upper bound on the average number of mistakes we make on our dataset. By minimizing the average hinge loss with $\mathbf{w}$, we can ensure that the SVM model makes few mistakes.

$$\frac{1}{N} M(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(y_i \neq \text{sgn}(\mathbf{w}^\top \mathbf{x}_i))$$

where $\mathbb{I}(y_i \neq \text{sgn}(\mathbf{w}^\top \mathbf{x}_i))$ is the indicator function that is 1 if $y_i \neq \text{sgn}(\mathbf{w}^\top \mathbf{x}_i)$ and 0 otherwise. The number of mistakes is the sum of the indicator functions. The hinge loss is:

$$\ell((\mathbf{x}, y), \mathbf{w}) = \max(0, 1 - y\mathbf{w}^\top \mathbf{x})$$

The hinge loss is non-negative, so the number of mistakes is less than or equal to the sum of the hinge losses:

$$\mathbb{I}(y_i \neq \text{sgn}(\mathbf{w}^\top \mathbf{x}_i)) \leq \max(0, 1 - y_i \mathbf{w}^\top \mathbf{x}_i)$$

Therefore, the average number of mistakes is less than or equal to the average hinge loss:

$$\frac{1}{N} M(\mathbf{w}) \leq \frac{1}{N} \sum_{i=1}^{N} \max\left(0, 1 - y_i \mathbf{w}^\top \mathbf{x}_i\right)$$

c. [3 points] Assume that the training data is separable. However, it turns out that the minimizer $\mathbf{w}^\star$ of the objective in Eq. (10) mis-classifies some training samples. How should we adjust λ (i.e., make it larger or smaller) to make the SVM work properly on training data? Why? Explain in 1-2 sentences.

We should adjust λ to make it smaller. A larger λ penalizes the norm of the weight vector more, which can lead to a simpler model. A smaller λ allows the model to fit the training data better, which can help reduce the number of mis-classified samples.

6 Support Vector Machine - Slack Variables & Duality Intuition

In hard SVM, our aim is to maximize the margin when samples are linearly separable. The margin is defined as the distance of the closest point to the classification boundary. In lecture, we showed that this is equivalent to solving the following hard-SVM optimization problem:

$$\min_{w,b} \frac{1}{2}\|w\|_2^2 \quad \text{s.t.} \quad y_i(w^\top x_i + b) \geq 1, \quad \forall i \in [n] \quad (12)$$

where $w, x_i \in \mathbb{R}^d$, $y_i \in \{-1, +1\}$, and n denotes the number of data points. Here, each (x_i, y_i) pair denotes a training data point.

a. [1 point] Explain in one or two sentences why the hard-SVM formulation given above may fail when applied to real-world datasets.

To address this issue, we use the soft-SVM formulation by introducing slack variables ξ_i for each data point $i = 1, 2, \ldots, n$:

$$\min_{w,b,\xi_1,\ldots,\xi_n} \left(\frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n} \xi_i \right)$$

$$\text{s.t.} \quad \forall i, \quad y_i(w^\top x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0.$$

The hard-SVM formulation may fail when applied to real-world datasets because the data may not be linearly separable. In such cases, the hard-SVM formulation will not find a solution that separates the classes without any errors.We use the soft-SVM formulation to allow for some mis-classifications in the training data.

b. [4 points] Discuss how the introduction of slack variables, ξ_i, resolves the impracticalities of the hard-SVM formulation that you identified in part (a). Why do we use "$\geq 1 - \xi_i$" instead of "≥ 1" in the constraints and why is it necessary to include "$C\sum_i \xi_i$" in the minimization objective? Your answer should be no more than three or four sentences long.

The introduction of slack variables, ξ_i, allows for some mis-classifications in the training data. The constraints are $y_i(w^\top x_i + b) \geq 1 - \xi_i$ instead of $y_i(w^\top x_i + b) \geq 1$ to allow for some margin of error. The term $C\sum_i \xi_i$ is included in the minimization objective to penalize the slack variables. The parameter C controls the trade-off between maximizing the margin and minimizing the number of mis-classifications.

c. [4 points] Further, as we see in the class, the dual formulation of the soft-SVM problem can be preferred.

Primal	Dual
$\min_{\boldsymbol{w},b,\xi_1,\ldots,\xi_n} \left\{ \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i \right\}$	$\max_{\boldsymbol{\alpha}} \left\{ \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \right\}$
s.t. $\quad \forall i,\ y_i(\boldsymbol{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0.$	s.t. $\quad \forall i,\ 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n}\alpha_i y_i = 0.$

where the dual variables are represented by α_i and $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$.

Figure 1: Primal Formulation of SVM

[3 points] Find the number of variables to be optimized in the primal and dual forms of SVM. Then, suppose that we decide to use some other kernels (e.g., radial basis kernel, polynomial kernel) instead of the linear kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$. Does this change the number of dual variables we need to use?

The primal problem:

$$\min_{w,b,\xi_1,\ldots,\xi_n} \left(\frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n} \xi_i \right)$$

Here, we directly optimize w and b and introduce slack variables ξ_i for each data point.

1. We have d features = d weights = d parameters
2. We have 1 bias term = 1 parameter
3. We have n slack variables (for each datapoint) = n parameters

The number of variables to be optimized in the primal form of SVM is $d + 1 + n$.

The Dual problem:

$$\max_{\alpha} \left(\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \right)$$

Here, we optimize the dual variables α instead of the primal variables w, b, ξ. We have:

1. We have n dual variables α_i (for each datapoint) = n parameters

The number of variables to be optimized in the primal form of SVM is $d + 1$ where d is the number of features. The number of variables to be optimized in the dual form of SVM is n where n is the number of data points. The number of dual variables does not change when using other kernels because the dual variables are the Lagrange multipliers associated with the constraints in the primal problem. The number of dual variables is determined by the number of data points, not the number of features or the kernel used.

d. [1 point] Why might you prefer to solve the SVM in the dual formulation as opposed to the primal formulation?

The number of variables to be optimized in the primal form of SVM is $d + 1$ where d is the number of features. The number of variables to be optimized in the dual form of SVM is n where n is the number of data points. The number of dual variables does not change when using other kernels. You might prefer to solve the SVM in the dual formulation because the dual formulation can be more computationally efficient when the number of data points is larger than the number of features.

7 Implementing SVMs

In this problem, you will experiment with SVMs on a real-world dataset. You will implement a linear SVM (i.e., an SVM using the original features). You should implement it from scratch without using any libraries like scikit-learn, although you can use existing packages to solve quadratic programs. If you are not sure if the library you want to use is allowed, please ask the TAs or post a question on Piazza. We provide you with a guided Python Notebook (hw2_svm_imp.ipynb) on which you will find parts you need to fill in. You can use this notebook on JupyterLab / JupyterNotebook / GoogleColab.

Please append your notebook file as a PDF at the end of your submission so that your code and cell outputs can be seen on the same submission PDF. If you use JupyterLab / JupyterNotebook / GoogleColab, you can save the notebook as PDF from File→Print. You can also use online 'ipynb to pdf' file converters. Please do not forget to append your notebook as a PDF to the end of your submission PDF for us to grade this question.

Dataset: We have provided the Heart Disease dataset from UCI's machine learning data repository. The provided binary classification dataset has 13 input features, and 303 samples.

- By running the first cell of the given notebook, you download the ucimlrepo package. It will help us download the dataset. For this step, you should have an internet connection.
- In the second cell, we provide all required libraries to complete the homework.
- In the third cell, we download the dataset (again, you need an internet connection), one-hot encode some categorical features, and split it into train/test parts using a set seed.

You should run those three cells without any modification. After running the first three cells, you should have `X_train` (shape of (216, 22)), `X_test` (shape of (83, 22)), `y_train` (shape of (216,)), and `y_test` (shape of (83,)) variables.

1. Let $x_k^{(1)}, \cdots, x_k^{(N)}$ be the values of feature k for all training set (X_{train}) points where N is the number of samples in the training set. Preprocess the training (X_{train}) and test data (X_{test}) by:

   (a) Computing the mean $\bar{x}_k = \frac{1}{N}\sum_{i=1}^{N} x_k^{(i)}$ of each feature and subtracting it from all values of this feature.

   (b) Dividing each feature by its standard deviation, defined as

   $$s_k = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_k^{(i)} - \bar{x}_k)^2}$$

   for feature k. Here, $\bar{x}_k$ is the sample mean of this feature.

   Save the normalized data samples as variables named `X_train_normalized` for the training set and `X_test_normalized` for the test set. Note that the shape of these variables should be the same as the shapes of `X_train` and `X_test`, respectively.

   This type of preprocessing is useful for SVMs, as SVMs attempt to maximize the distance between the separating hyperplane and the support vectors. If one feature (i.e., one dimension in this space) has very large values, it will dominate the other features when calculating this distance. Rescaling the features will ensure that they all have the same influence on the distance metric.

   Note that the mean and standard deviation should be estimated from the training data (`X_train`) and then applied to both datasets. This is because using the statistics from the training data ensures that the model is not biased by the test data, which should remain unseen during training.

   Report the mean and the standard deviation of the first and the last features computed on the training data (`X_train`):

   - Mean of the first feature: $\bar{x}_1$
   - Standard deviation of the first feature: s_1
   - Mean of the last feature: $\bar{x}_{22}$
   - Standard deviation of the last feature: s_{22}

2. Now, implement the `trainSVM` function. Follow the description given in the cell with the title `Train SVM`. The input of `trainSVM` contains normalized training data (`X_train_normalized`) and training labels (`y_train`), as well as the parameter C. The output of `trainSVM` contains the optimal weight vector (w), bias term (b) and slack variables (ξ). In your implementation, solve the SVM in its primal form:

   $$\min_{w,b,\xi} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{N}\xi_i$$
   $$\text{s.t.} \quad y_i(w^\top x_i + b) \geq 1 - \xi_i, \quad \forall 1 \leq i \leq N$$
   $$\xi_i \geq 0, \quad \forall 1 \leq i \leq N$$

   To solve the above quadratic problem, we encourage you to use the `CVXPY` Python package. You can also use any package of your choice. If you are not familiar with `CVXPY`, they have their own very extensive tutorial/user guide at: `https://www.cvxpy.org/tutorial/index.html`. After implementing the desired function,

   - Solve the problem using `trainSVM` with normalized training set (`X_train_normalized`), training labels (`y_train`) and $C = 1$. What values do you find for the first 3 weights $\{w_1, w_2, w_3\}$, b, and first 3 slack variables $\{\xi_1, \xi_2, \xi_3\}$?
   - Solve the problem using `trainSVM` with normalized training set (`X_train_normalized`), training labels (`y_train`) and $C = 0$. What values do you find for the first 3 weights $\{w_1, w_2, w_3\}$, b, and first 3 slack variables $\{\xi_1, \xi_2, \xi_3\}$?
   - Do you see any difference between the slack variables you found with $C = 0$ and $C = 1$? Does that explain why we are introducing $C\sum_i \xi_i$ term into the objective for Soft-SVM? Explain your answer in 1-2 sentences.

3. Now, we will write a function for evaluation in the cell titled `Eval SVM`. Write a function `evalSVM` that takes four inputs: `X_eval` - features for evaluation (we will use either normalized train set features (`X_train_normalized`) or normalized test set features (`X_test_normalized`) to calculate train/test set accuracy, respectively), `y_eval` - true labels of the evaluation data (this will be again used to input train set labels (`y_train`) or test set labels (`y_test`)), and the trained SVM parameters `w` and `b`. This function should estimate the classes of all samples in the given `X_eval` for evaluation using an SVM model with parameters `w` and `b`. Then, the function should compare the estimated labels with `y_eval` (the true labels), and return the accuracy of the SVM model on this data, which is the fraction of data points that are correctly classified.

   Now, for all possible $C = a \times 10^q$ values where $a \in \{1, 3, 6\}$ and $q \in \{-4, -3, -2, -1, 0, 1\}$, find the optimal values of `w` and `b` using `trainSVM` with training labels (`y_train`) and normalized training set (`X_train_normalized`) and then, using your `evalSVM` function, calculate the accuracy over the normalized training set (`X_train_normalized`) and normalized test set (`X_test_normalized`).

   Make two plots of your results, one showing the training accuracy vs. the value of C and one showing the test accuracy vs. the value of C. Use a log-scale on the x-axis for both plots. Also, report the values of C that maximize the training and test accuracies, along with the accuracies that are reached with these values of C.

Deliverables

Normalisation of `X_train` and `X_test` [3 points]

- Correct implementation of data normalisation on `X_train` and `X_test`.
- Explanation of why the mean and standard deviation used for normalising both training and test datasets are estimated from the training data.
- Report of the correct mean and standard deviation of the first and last features of the training data.

SVM training [7 points]

- Correct implementation of the `trainSVM` function.
- Report of the first three weights, bias, and first three slack variables for both $C = 1$ and $C = 0$.
- Identification of the difference between the slack variables when $C = 1$ and $C = 0$. And, an explanation of the connection between this difference and the $C \sum_i \xi_i$ term in the Soft-SVM objective.

SVM evaluation [5 points]

- Correct implementation of the `evalSVM` function.
- Plot of training accuracy against the instructed values of C.
- Plot of test accuracy against the instructed values of C.
- Report of the value of C that maximises training accuracy and the corresponding training accuracy.
- Report of the value of C that maximises test accuracy and the corresponding test accuracy.

8 Non-linear Basis Functions [20 points]

8.1 Derivation [6 points]

In class, we have learned that by using non-linear basis functions, we can fit a non-linear model (w.r.t. to the data) while preserving linearity w.r.t. the model parameters. In this problem, we are given a training data set D that consists of N points, (x_i, y_i) for $i = 1, \cdots, N$, and we consider a regression problem as follows:

$$y_i = w_0 + w_1 \sin(x_i) + w_2 \cos(x_i) + w_3 \sin(2x_i) + w_4 \cos(2x_i) + \cdots + w_{2k-1} \sin(kx_i) + w_{2k} \cos(kx_i)$$

Here x_i and y_i are scalar real numbers.

1. [2 points] What is the basis function $\phi(x)$ for this problem?

$$\phi(x) = [1, \sin(x), \cos(x), \sin(2x), \cos(2x), \cdots, \sin(kx), \cos(kx)]^\top$$

2. [2 points] Express the residual sum of squares error in terms of $\{\phi(x_i); i = 1, 2, \ldots, N\}$.

$$\text{RSS} = \sum_{i=1}^{N} (y_i - \phi(x_i)^\top \mathbf{w})^2$$

3. [2 points] What parameter values $\mathbf{w} = [w_0, w_1, \cdots, w_{2k}]^\top$ minimize the residual sum of squares error?

   The parameter values that minimize the residual sum of squares error are:

$$\mathbf{w} = (\Phi^\top \Phi)^{-1} \Phi^\top \mathbf{y}$$

   where Φ is the matrix whose ith row is $\phi(x_i)$ and $\mathbf{y}$ is the vector of target values.

8.2 Implementation

Now we want to implement sinusoidal regression on the $N = 61$ data points given to you in `nonlinear-regression-data.csv`. In `nonlinear-regression.py`, implement the following functions:

(a) `phi(x)`: which returns the basis function you got in the derivation part.

(b) `fit(X_train, Y_train, k)`: which fits the nonlinear model given the degree k.

(c) `predict (X)`: which predicts the values of the dependent variable for a new set of covariates using the learned model.

(d) `rmse (X_val, Y_val)`: which computes the RMSE error on the validation set using the learned model.

8.3 Evaluation [14 points]

Now, randomly divide the dataset into two parts: training set (45 points) and validation set (16 points). By varying k from 1 to 10, obtain the RMSE error on the validation set and the training set.

1. [4 points] Plot the training error (RMSE error on the training set) versus k. From the plot, which value of k gives you the minimum training error?
2. [4 points] Plot the validation error versus k. Which value of k gives you the minimum validation error?
3. [2 points] Compare the optimal k values that you found for the training error and validation error. Explain why they are different or the same.
4. [4 points] For each $k = 1, 3, 5, 10$, create a scatter plot of the validation data points. On the same plot, for each k, draw a line for the fitted polynomial.

Please do not forget to append your Python code and notebook to the end of your submission for us to grade this question.

svm

February 21, 2025

[1]:
```
!pip install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in
/home/kip/projects/MathsRequiredForAI/EnvMaths/lib/python3.12/site-packages
(0.0.7)
Requirement already satisfied: pandas>=1.0.0 in
/home/kip/projects/MathsRequiredForAI/EnvMaths/lib/python3.12/site-packages
(from ucimlrepo) (2.2.3)
Requirement already satisfied: certifi>=2020.12.5 in
/home/kip/projects/MathsRequiredForAI/EnvMaths/lib/python3.12/site-packages
(from ucimlrepo) (2024.8.30)
Requirement already satisfied: numpy>=1.26.0 in
/home/kip/projects/MathsRequiredForAI/EnvMaths/lib/python3.12/site-packages
(from pandas>=1.0.0->ucimlrepo) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/home/kip/projects/MathsRequiredForAI/EnvMaths/lib/python3.12/site-packages
(from pandas>=1.0.0->ucimlrepo) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/home/kip/projects/MathsRequiredForAI/EnvMaths/lib/python3.12/site-packages
(from pandas>=1.0.0->ucimlrepo) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/home/kip/projects/MathsRequiredForAI/EnvMaths/lib/python3.12/site-packages
(from pandas>=1.0.0->ucimlrepo) (2024.2)
Requirement already satisfied: six>=1.5 in
/home/kip/projects/MathsRequiredForAI/EnvMaths/lib/python3.12/site-packages
(from python-dateutil>=2.8.2->pandas>=1.0.0->ucimlrepo) (1.16.0)
```

[2]:
```
from ucimlrepo import fetch_ucirepo
import pandas as pd
import numpy as np
import cvxpy as cp
from matplotlib import pyplot as plt
```

[3]:
```
# !! DO NOT MODIFY THIS CELL !!

# Download and preprocess the dataset.
# fetch dataset
heart_disease = fetch_ucirepo(id=45)
```

```
X = heart_disease.data.features
# Convert categorical features into one-hot encode
categorical_features = ['cp','thal','slope','restecg']
X = pd.get_dummies(X, columns=categorical_features)

y = heart_disease.data.targets
print(f"Number of samples in all full dataset is: {len(X)}.")

# Check if our train set has missing value
na_in_features = X.isna().any(axis=1).sum()
na_in_trainY = y.isna().sum()
print(f"Number of rows with missing values in features: {na_in_features}")

# Drop the rows with missing values.
indices_with_nan = X.index[X.isna().any(axis=1)]
X = X.drop(indices_with_nan)
y = y.drop(indices_with_nan)

# Divide train/test
np.random.seed(6464)
msk = np.random.rand(len(X)) < 0.75
X_train = X[msk]
X_test = X[~msk]
y_train = y[msk]
y_test = y[~msk]

# Convert problem to binary problem
X_train = np.array(X_train,dtype='float')
X_test = np.array(X_test,dtype='float')
y_train = np.array([-1 if i==0 else 1 for i in y_train.values],dtype='float')
y_test = np.array([-1 if i==0 else 1 for i in y_test.values],dtype='float')

print(f"Shapes: X_train: {X_train.shape}, y_train: {y_train.shape}, X_test:␣
↪{X_test.shape}, y_test: {y_test.shape}")
```

```
Number of samples in all full dataset is: 303.
Number of rows with missing values in features: 4
Shapes: X_train: (216, 22), y_train: (216,), X_test: (83, 22), y_test: (83,)
```

[4]:
```
# Normalize X_train and X_test using the statistics of X_train.
# 1. Compute the mean and standard deviation for each feature in X_train
# 2. Subtract the mean from each feature and divide by the standard deviation
#    for both X_train and X_test.

mean = np.mean(X_train,axis=0)
std = np.std(X_train,axis=0)
X_train_normalized = (X_train-mean)/std
```

```
X_test_normalized = (X_test-mean)/std

print(f"Shapes: X_train: {X_train_normalized.shape}, y_train: {y_train.shape},␣
 ↪X_test: {X_test_normalized.shape}, y_test: {y_test.shape}")
```

```
Shapes: X_train: (216, 22), y_train: (216,), X_test: (83, 22), y_test: (83,)
```

[5]:
```
# Print the mean and standard deviation of the first and last feature.

print(f"Mean of the first feature: {np.mean(X_train_normalized[:, 0])}")
print(f"Standard deviation of the first feature: {np.std(X_train_normalized[:,␣
 ↪0])}")
print(f"Mean of the last feature: {np.mean(X_train_normalized[:, -1])}")
print(f"Standard deviation of the last feature: {np.std(X_train_normalized[:,␣
 ↪-1])}")
```

```
Mean of the first feature: 4.11193712824132e-17
Standard deviation of the first feature: 0.9999999999999999
Mean of the last feature: -5.756711979537848e-17
Standard deviation of the last feature: 1.0
```

0.0.1 Explanation of Normalization Using Training Data Statistics

The mean and standard deviation for normalization are estimated from the training data to prevent **data leakage** and ensure that the model generalizes well to unseen data. Using separate statistics for `X_test` could lead to inconsistent feature scaling, making model predictions unreliable.

0.0.2 TRAIN SVM

[6]:
```
# Train SVM

# Complete the `trainSVM` function to find the optimal w and b that minimize
# the primal SVM objective given in the write-up.
# The function takes three inputs:
# - trainX: the normalized train features with shape (#train_samples, #features)
# - trainY: train labels with shape (#train_samples,)
# - C: C parameter of the minimization problem
# The function should return a three-tuple with:
# - w: the weight vector with shape (#features,)
# - b: the bias. A scalar with shape (1,)
# - xi: the slack variables with shape (#train_samples,)

# You can use cvxpy that we imported as cp
# You may find cp.Variable, cp.Minimize, cp.Problem useful
# For the problem solver, prefer the default, cp.CLARABEL

def trainSVM(trainX, trainY, C):
    w = cp.Variable(trainX.shape[1])
```

```
    b = cp.Variable()
    xi = cp.Variable(trainX.shape[0])

    objective = cp.Minimize(0.5*cp.norm(w,2)**2 + C*cp.sum(xi))
    constraints = [cp.multiply(trainY,(trainX@w+b)) >= 1-xi, xi >= 0]
    prob = cp.Problem(objective, constraints)
    prob.solve()
    return w.value, b.value, xi
```

[7]:
```
# Solve SVM with C = 1 and print the first three weights, b and the first
# three slack variables as instructed in the write-up

Y = y_train
C = 1
w, b, xi = trainSVM(X_train_normalized, Y, C)
print(f"First three weights: {w[:3]}")
print(f"b: {b}")
print(f"First three slack variables: {xi.value[:3]}")
```

```
First three weights: [-0.01280084  0.51706872  0.27813637]
b: 0.08109278708401382
First three slack variables: [-1.70119328e-10 -1.64395885e-10 -1.69587409e-10]
```

[8]:
```
# Solve SVM with C = 0 and print the first three weights, b and the first
# three slack variables as instructed in the write-up

C = 0
w, b, xi = trainSVM(X_train_normalized, Y, C)
print(f"First three weights: {w[:3]}")
print(f"b: {b}")
print(f"Actual first three slack variables: {xi.value[:3]}")
```

```
First three weights: [ 3.09523259e-06 -8.18802636e-06 -9.46615246e-06]
b: -10.447621082517728
Actual first three slack variables: [429.58840105 434.02071004 414.34670026]
```

0.0.3 Difference Between the Slack Variables When C = 1 and C = 0

When we solve the SVM with different values of C, the slack variables ( ) change. The slack variables represent the degree to which each data point violates the margin.

For C = 1: - The first three slack variables are: [-1.70119246e-10 -1.64395808e-10 -1.69587327e-10]

For C = 0: - The first three slack variables are: [429.58840105 434.02071004 414.34670026]

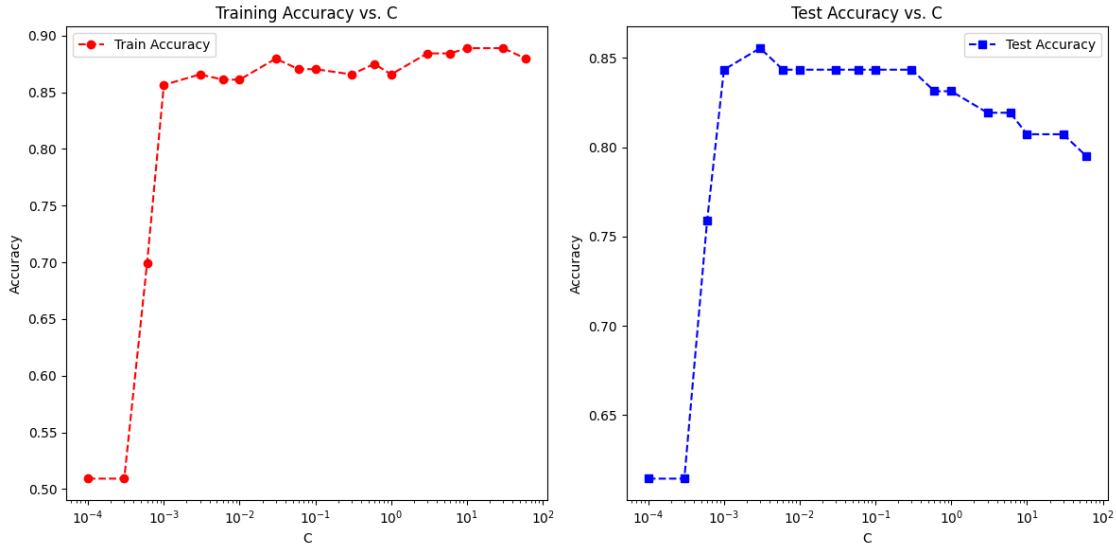
The slack variables x_i will be larger when C = 0 compared to C = 1. This is because with a smaller C, the model has less penalty for misclassification because it encourages large margins, allowing more slack (larger x_i) for violations of the margin. As C increases it penalizes the misclassifications. It encourages the correct classification over a wider margin, enventually with a large c it will approach SVM hard margin

[9]:
```
# Eval SVM

# Write a function to evaluate the SVM model given its `w` and `b` parameters
# on evaluation data `X_eval` and true labels `y_eval`.
#  1. Estimate the labels of `X_eval`.
#  2. Return the ratio of accurately estimated labels by comparing with␣
 ↪`y_eval`.


def evalSVM(X_eval, y_eval, w, b):
    y_pred = np.sign(X_eval@w + b)
    return np.mean(y_pred == y_eval)

print(f"Accuracy on the test set: {evalSVM(X_test_normalized, y_test, w, b):.
 ↪2f}")
```

```
Accuracy on the test set: 0.61
```

[18]:
```
# Define the range of C values
C_values = [a * 10**q for a in [1, 3, 6] for q in [-4, -3, -2, -1, 0, 1]]
C_values = sorted(C_values)

train_accuracies = []
test_accuracies = []

# Calculate the optimal w, b and accuracies for each C value
for C in C_values:
    w, b, _ = trainSVM(X_train_normalized, y_train, C)
    train_acc = evalSVM(X_train_normalized, y_train, w, b)
    test_acc = evalSVM(X_test_normalized, y_test, w, b)
    train_accuracies.append(train_acc)
    test_accuracies.append(test_acc)

# Plot the training accuracy vs. the value of C
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(C_values, train_accuracies, label='Train Accuracy', color='red',␣
 ↪linestyle='dashed', marker='o')
plt.xscale('log')
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title('Training Accuracy vs. C')
plt.legend()

# Plot the test accuracy vs. the value of C
plt.subplot(1, 2, 2)
```

```
plt.plot(C_values, test_accuracies, label='Test Accuracy', color='blue',␣
 ↪linestyle='dashed', marker='s')
plt.xscale('log')
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title('Test Accuracy vs. C')
plt.legend()

plt.tight_layout()
plt.show()

# Report the values of C that maximize the training and test accuracies
max_train_acc_index = np.argmax(train_accuracies)
max_train_acc_C = C_values[max_train_acc_index]
max_train_acc = train_accuracies[max_train_acc_index]

max_test_acc_index = np.argmax(test_accuracies)
max_test_acc_C = C_values[max_test_acc_index]
max_test_acc = test_accuracies[max_test_acc_index]

print(f"The value of C that maximizes training accuracy is: {max_train_acc_C}")
print(f"The corresponding training accuracy is: {max_train_acc}")

print(f"The value of C that maximizes test accuracy is: {max_test_acc_C}")
print(f"The corresponding test accuracy is: {max_test_acc}")
```




```
The value of C that maximizes training accuracy is: 10
The corresponding training accuracy is: 0.8888888888888888
The value of C that maximizes test accuracy is: 0.003
```

```
The corresponding test accuracy is: 0.8554216867469879
```

nonlinear-regression

February 21, 2025

0.1 IMPORTS

[93]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from numpy.linalg import pinv
from sklearn.model_selection import train_test_split
```

0.2 sinusoidal regressor

[94]:
```
class SinusoidalRegressor:
    def __init__(self):
        self.k = None
        self.weights = None

    def phi(self, x):
        """Generate basis functions for a given x"""
        return np.column_stack((np.ones_like(x := np.asarray(x)), np.sin(np.
 ↪outer(x, np.arange(1, self.k + 1))), np.cos(np.outer(x, np.arange(1, self.k␣
 ↪+ 1)))))

    def fit(self, X_train, Y_train, k):
        """Train the model using pseudo-inverse with optional regularization"""
        self.k = k
        X_train = X_train
        Y_train = Y_train

        Phi = self.phi(X_train)
        self.weights = pinv(Phi.T @ Phi) @ Phi.T @ Y_train

    def predict(self, X):
        """Make predictions"""
        if self.weights is None:
            raise ValueError("Model is not fitted yet.")
        X = X
        Phi = self.phi(X)
        return Phi @ self.weights
```

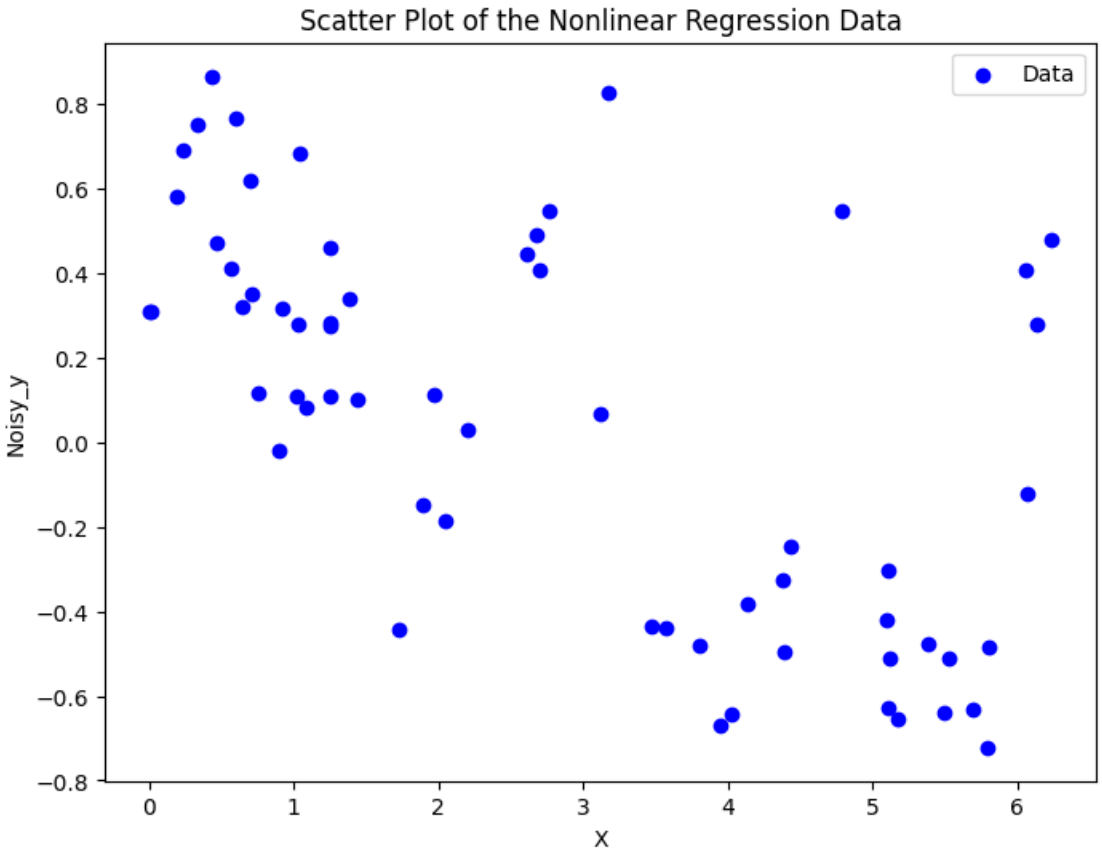
```
    def rmse(self, X_val, Y_val):
        """Compute RMSE"""
        y_pred = self.predict(X_val)
        return np.sqrt(np.mean((Y_val - y_pred) ** 2))
```

0.3 Loading the data and the model

[95]:
```
# Load Data
np.random.seed(61)
csv_file = 'nonlinear-regression-data.csv'
data = pd.read_csv(csv_file)
x = np.array(data['X'])
y = np.array(data['Noisy_y'])

# Split Data
X_train, X_val, Y_train, Y_val = train_test_split(x, y, test_size=16,␣
 ↪random_state=61)

# Initialize Model
sinusoidal_regressor = SinusoidalRegressor()
```

VISUALIZING THE DATASET We have an independent variable X and some dependent Noisy_Y that is associated with each value of X.

[96]:
```
# Plot the data
plt.figure(figsize=(8, 6))
plt.scatter(x, y, color='blue', label='Data')
plt.xlabel('X')
plt.ylabel('Noisy_y')
plt.title('Scatter Plot of the Nonlinear Regression Data')
plt.legend()
plt.show()

# size of the dataset
N = len(X_train)
print(f"Size of the dataset: {N}")
N = len(X_val)
print(f"Size of the dataset: {N}")
```



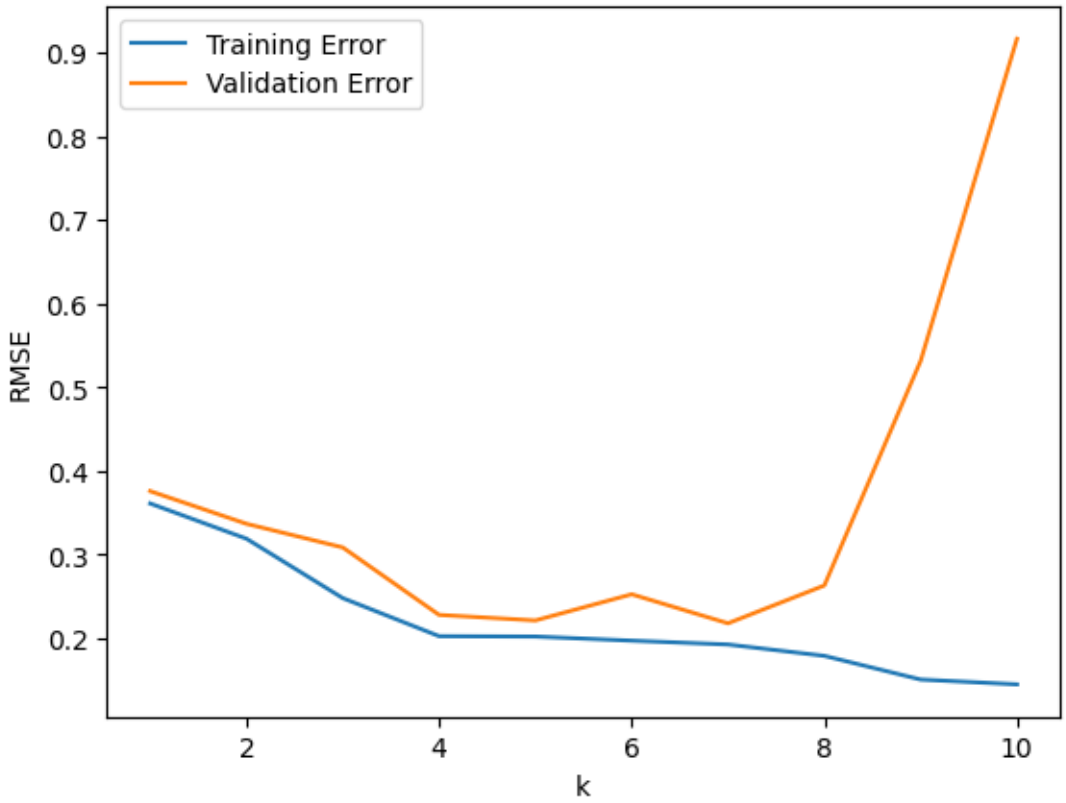

```
Size of the dataset: 45
Size of the dataset: 16
```

0.4 TRAINING THE MODEL

As K increases, we increase more sine and cosine basis functions used in the model, this makes the model more complex.

[97]:
```python
# Fit model and compute errors
train_errors = []
val_errors = []
k_values = range(1, 11)
for k in k_values:
    sinusoidal_regressor.fit(X_train, Y_train, k)
    train_errors.append(sinusoidal_regressor.rmse(X_train, Y_train))
    val_errors.append(sinusoidal_regressor.rmse(X_val, Y_val))

# Plot Training and Validation Errors
plt.plot(k_values, train_errors, label="Training Error")
plt.plot(k_values, val_errors, label="Validation Error")
```

```
plt.xlabel("k")
plt.ylabel("RMSE")
plt.legend()
plt.show()
```



[98]:
```
optimal_k_train = np.argmin(train_errors) + 1
print(f"The optimal k that minimizes the training data is: {optimal_k_train}")
```

```
The optimal k that minimizes the training data is: 10
```

[99]:
```
optimal_k_val = np.argmin(val_errors) + 1
print(f"The optimal k that minimizes the validation error is: {optimal_k_val}")
```

```
The optimal k that minimizes the validation error is: 7
```

0.4.1 Compare the optimal k values that you found for the training error and validation error. Explain why they are different or the same.

The optimal k that minimizes the training error is: 10

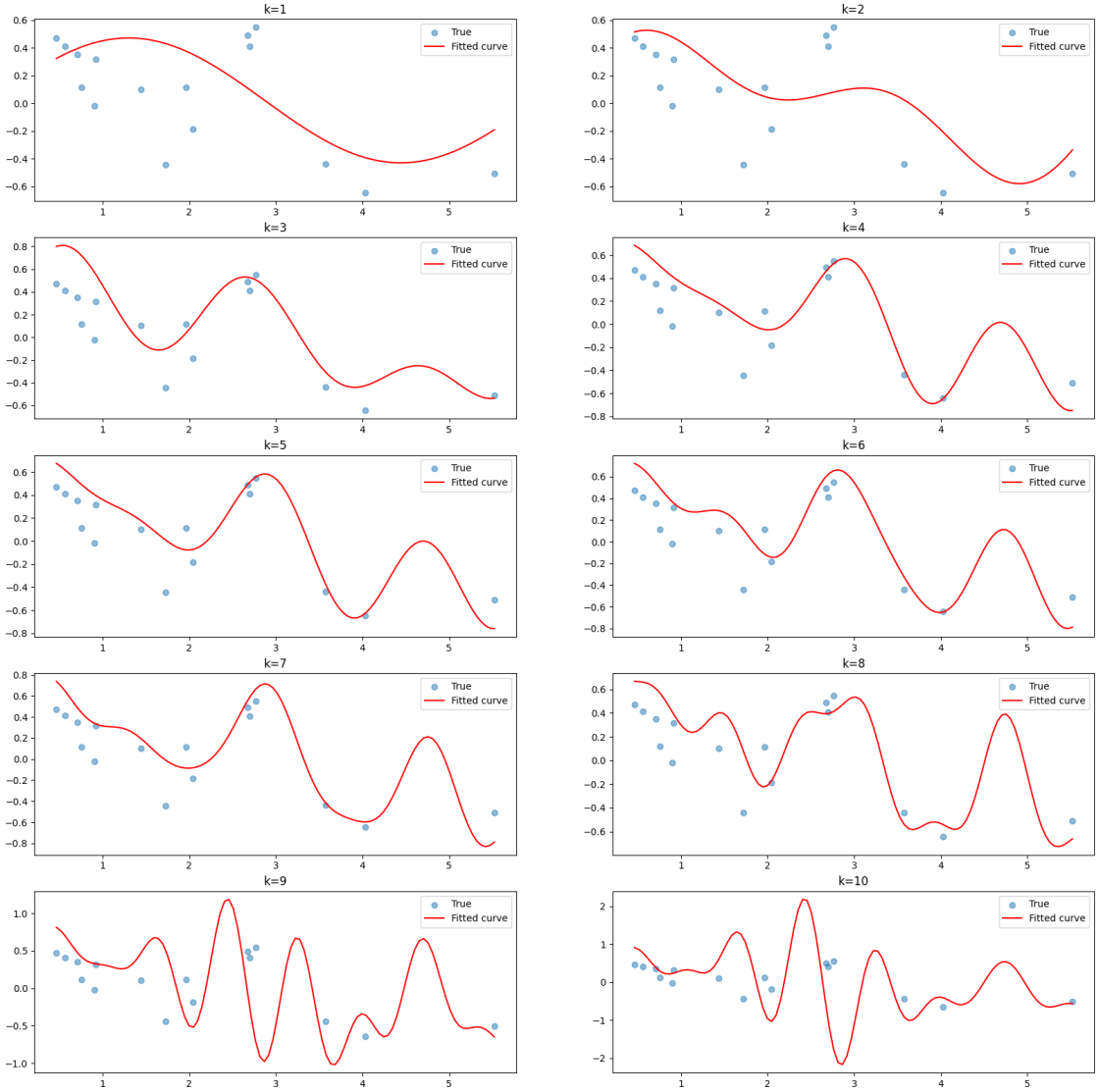
The optimal k that minimizes the validation error is: 7

The optimal k values for training and validation errors are different. This discrepancy occurs

because the training error measures how well the model fits the training data, while the validation error measures how well the model generalizes to unseen data. A lower k value might fit the training data very well (low training error) but may not generalize well to new data (high validation error). Conversely, a higher k value might provide a better balance between fitting the training data and generalizing to new data, resulting in a lower validation error. This phenomenon is known as overfitting and underfitting. The optimal k for validation error is typically preferred as it indicates better generalization performance.

0.5 MODEL PREDICTIONS

[100]:
```
# Plot Predictions for Different k Values
fig, axs = plt.subplots(5, 2, figsize=(20, 20))
x_line = np.linspace(X_val.min(), X_val.max(), 100)
for k in k_values:
    sinusoidal_regressor.fit(X_train, Y_train, k)
    y_line = sinusoidal_regressor.predict(x_line)

    ax = axs[(k - 1) // 2, (k - 1) % 2]
    ax.scatter(X_val, Y_val, label="True", alpha=0.5)
    ax.plot(x_line, y_line, color='r', label="Fitted curve")
    ax.set_title(f"k={k}")
    ax.legend()
plt.show()
```



0.5.1 For each k = 1, 3, 5, 10, create a scatter plot of the validation data points. On the same plot, for each k, draw a line for the fitted polynomial.

[101]:
```
# Plot Predictions for k = 1, 3, 5, 10
k_values_to_plot = [1, 3, 5, 10]
fig, axs = plt.subplots(2, 2, figsize=(15, 10))

for idx, k in enumerate(k_values_to_plot):
    sinusoidal_regressor.fit(X_train, Y_train, k)
    y_line = sinusoidal_regressor.predict(x_line)

    ax = axs[idx // 2, idx % 2]
    ax.scatter(X_val, Y_val, label="Validation Data", alpha=0.5)
```

```
    ax.plot(x_line, y_line, color='r', label=f"Fitted curve (k={k})")
    ax.set_title(f"k={k}")
    ax.legend()

plt.tight_layout()
plt.show()
```