Statistical learning theory

1. **Learning from examples:**
   - Machine learning involves training from examples rather than explicit programming
   - The framework focuses on supervised learning where the goal is to map the input data x into outputs y.

2. **Supervised learning:**
   - A training set $S = \{(x_1, y_1), \ldots (x_n, y_n)\}$
   - Goal: to find an input-output relation $f(x)$ where $f(x_{new}) \cong y$

3. **Data space and distribution:**
   - Input space $X \subseteq R^D$ output space $Y \subseteq R$.
   - Data modeled by joint distribution $p(x, y)$ often factorized as $p(x, y) = p_X(x)p(y|x)$.

4. **Loss function:**
   - A function that evaluates how much we lose if instead of real outputs we use the predicted outputs.
   - $l\left(y_j, f(x_j)\right) = (y_j - f(x_j))^2$

5. **Expected Loss (or expected risk):**
   - Is defined as the average loss over all possible $P(x, y)$ for every input-out pair, weighted by their probability of distribution.
   - $\varepsilon(f) = E[l(y, f(x)] = \int p(x, y) l(y, f(x)) dxdy$

6. **Learning Algorithms and Generalization:**
   - A learning algorithm computes an estimator f_s from training data S.
   - Generalization: f_s should perform well on unseen data.
   - Ecess Expected Risk: $E(f_s) - E(f_s^*)$ Ideally minimized

7. **Consistency:**
   - A learning algorithm is consistent, if as the number of training sample m approaches infinity, the expected risk between empirical and ideal model reaches zero.
   - $\lim_{m \to \infty} E\left[\varepsilon(f_m) - \varepsilon(f^*)\right] = 0$

8. **Fitting and stability:**
   - Fitting: The model should fit the training data well.
   - Stability: The model shouldn't change drastically with slight variations of data.
   - Overfitting: The model is too sensitive to noise (good fit, low stability)
   - Oversmoothing: The model is bias (High stability, poor fit)

9. **Regularization:**
   a. Balance fitting and stability
   b. Sample Complexity: the minimum data size $n(\epsilon)$ needed to achieve error bound ε

Sample Questions:

1. **What is the primary goal of supervised learning?**
   The primary goal of supervised learning is to find an input-output relationship $f(x)$ such that $f(x_{new}) \cong y$ for unseen data, based on the given training set $S = \{(x_1, y_1), \dots (x_n, y_n)\}$

2. **Why is it necessary to model the data distribution $p(x, y)$ in statistical learning?**
   Modeling $p(x, y)$ accounts for uncertainty in data. The joint distribution factorizes as $p(x, y) = p_X(x)p(y|x)$, where $p_X(x)$ represents the input space variability and $p(y|x)$ accounts for noise in the outputs.

3. **Explain the significance of lost function $l\left(y_j, f\left(x_j\right)\right)$ in statistical learning.**
   The loss function quantifies the cost of predicting $f(x)$ instead of the true value y. It guides learning by providing a measure of error that the learning algorithm minimizes to estimate the target function.

4. **Derive the expected risk $E(f)$ and explain its components.**

   $$E(f) = E[l(y, f(x)] = \int p(x, y)l\left(y, f(x)\right)dxdy$$

   - $p(x, y)$: joint distribution of data
   - $l(y, f(x))$: Point-wise error measure
   - $E(f)$: Averages the error over all possible pairs, capturing both past and feature error

5. **Why is the target function f^* uncomputable in practice?**
   The target function f^* minimizes the expected risk $E(f)$, but it depends on the true data distribution $p(x, y)$, which is unknown in practice.

6. **What's the role of training set S in training?**
   A training set S provides samples (x_i, y_i) from $p(x, y)$. A learning algorithm uses S to compute an estimation f_S that approximation the target function f^*.

7. **Define and interpret the excess expected risk $E(f_S) - E(f^*)$.**
   Excess expected risk quantifies the difference between the expected risk of the estimation f_S and the target function f^*. It measures how much worse f_S performs compared to the optimal model f^*.

8. **What's the definition of consistency in learning algorithms?**
   A learning algorithm is consistent if the sample size $n \to \infty$, the expected excess risk $E_S[E(f_S) - E(f^*)] \to 0$. This implies the algorithm performance converges to the optimal model with sufficient data.

9. **How does regularization help balance fitting and stability?**
   Regularization introduces a parameter the penalizes overfitting while ensuring the model remains stable. It prevents the model from fitting noise in the data while controlling the complexity of f_s.

10. **What is sample complexity and why is it important?**
    A sample complexity $n(\epsilon)$ is the minimum number of samples required to achieve an error bound ϵ. It provides insight on how much data is needed for reliable learning.

11. **Differentiate between overfitting and oversmoothing.**
    - Overfitting: the model fits the training data too well but lack stability, leading to poor generalization.
    - Oversmoothing: The model prioritizes stability but fails to capture important features, resulting in underfitting.

12. **Why f_S considered random, how does this affect generalization?**
    f_S depends on specific training set S, which is a random sample from $p(x, y)$. This randomness introduces variability in f_S, affecting its ability to generalize to unseen data.

13. **What does the noiseless classification assumption $p(1|x) = 1 - p(-1|x)$ imply?**

In noiseless classification, for any input $x, p(1|x)$ is either 1 or 0. This means there is no uncertainty in the output: the model is completely confident about class assignment.

Local methods

1. Learning from local methods

- Nearest neighbors (NN):
  - Predict y^* for a new point x^* based on the nearest neighbor in $S = \{(x_i, y_i)\}$
  - $y^* = y_i, where\ j = argmin_i\ |x^* - x|$
  - Computational cost $O(nD)$ where n is the dataset size and D is the dimensionality.
- K-nearest neighbors (KNN):
  - Predict y^* as the average of K nearest neighbors.
  - $y^* = \frac{1}{K} \Sigma y_i$
  - Computational cost $O(nD + nlogn)$
  - It's a flexible algorithm if it's provided enough data. However, its nervous algorithm. Meaning its sensitive to small changes in data. That's because it relies on distance between points to make prediction.
- Parzen window:
  - Assign weights to neighbors based on proximity using a similarity function $k(x, x')$.
  - $\hat{f}(x) = \frac{\Sigma_{i=1}^{n} y_i k(x,x')}{\Sigma_{i=1}^{n} k(x,x')}$
  - Gaussian kernel, linear kernel, linear decay, quadric decay

2. Bias-Variance Trade-off

- Bias:
  - Difference between true function $f^*(x)$ and the expected prediction.
  - High bias -> Underfitting
- Variance:
  - Sensitivity of the model to the changes in training data.
  - High variance -> Overfitting
- Tradeoff:
  - Small K: low bias, high variance
  - Large K: high bias, low variance
  - Optimal $K: Minimize\ total\ error\ E_k(x) = bias^2 + variance + irreducible\ noise(\sigma^2)$

3. Hyperparameter selection:

- Optimal hyperparameter (K^*):
  - Minimizes the expected loss: $E_K = E_S E_{x,y}[(y - \hat{f}_{S,K}(x))^2]$
  - $K^* = argmin\ \varepsilon_K$
  - In practice the true distribution is unknown
  - The optimal K minimizes the sum of bias and variance, balancing model's complexity.
  - Small dataset or noisy data: use large K to reduce variance.
  - Large clean dataset: smaller K can capture finer details.
- Hold-out Cross validation:
  - Split the data into training (T) and validation (V) sets.
  - Train on T and compute the loss $\hat{E}_k$ on (V).
  - Select K that minimizes the $\hat{E}_k$.
- V-fold cross validation:
  - Divide the data into V non-overlapping folds.
  - Use V-1 folds for training and the remaining fold for validation.

- Repeat for all V folds using each fold once as the validation.
- Average the validation errors across all folds.

- Leave on out cross validation:
  - It's the extreme version of V-fold where $V = n$ meaning we iterate through all the data points. Use all but one data point for training, validate on the excluded point.

4. **Training vs. Validation Error:**
   - Training Error:
     - Decreases as K increases.
   - Validation Error:
     - Follows a U-shape curve, reflecting underfitting and overfitting.
     - Optimal K minimizes validation error.

5. **Regression Setting:**
   - Noisy data model: $y = f^*(x) + \delta$, where δ is noise ($E[\delta] = 0, Var(\delta) = \sigma^2$).
   - Total error $E_K(x): bias^2 + variance + \sigma^2$.

Sample Questions:

1. **Define a local learning method and provide an example.**
   Local learning methods make prediction based on near points rather than global model.
   Example: K-Nearest Neighbors (KNN) which assign the average values of K nearest neighbors of a new data point (regression) or based on the majority of classes (Classification).

2. **What is the difference between KNN and NN?**
   - NN assigns the value of the closest training sample.
   - KNN takes the majority class (Classification) or mean of K-nearest neighbors (Regression). It's less sensitive compared to NN.

3. **Why is the choice of distance metric important in KNN?**
   The choice of metrics determines how closeness is measured between points.
   - Euclidian distance: Suitable for continues numerical data.
   - Manhattan distance: works well for high-dimensional sparse data.
   - Hamming distance: used for categorical or binary data.

4. **Explain the computational complexity of KNN.**
   - Brute force of $O(nD)$ for computing the distance and $O(nlogn)$ for sorting leading to
   - $O(nD + nlogn)$

5. **What is Parzen window, and how does it improve KNN?**
   - Parzen window generalizes KNN by introducing a kernel function $K(x, x')$ that assigns weights to the neighbors based on their distances.
   - Closer points have more influence over predictions.

6. **Name and describe two common kernel functions in Parzen windows.**
   - Gaussian kernel: $K(x, x') = e^{-\frac{||x-x'||^2}{2\sigma}}$.
   - Smoothly decreases with distance.
   - Linear Kernel: $k(x, x') = (1 - \frac{||x-x'||}{\sigma}$
   - Weights decrease linearly with distance.

7. **What's bias variance tradeoff?**
   - The total error is decomposed into: $Error\ = Bias^2 + Variance + Irreducible\ Error$
   - High bias: underfitting, over simplified model (large K in KNN)
   - High variance: overfitting, model captures noise (small K in KNN)

8. **How does KNN's choice of K affect the bias-variance tradeoff?**
   - Small K: low bias, high variance (overfitting)
   - Large K: high bias, low variance (underfitting)

9. **What is the irreducible error in bias-variance tradeoff?**
   - It is the inherent noise in the data, denoted as σ^2, which no model can eliminate.

10. **Derive the formula for KNN regression prediction:**

$$\hat{f}(x^*) = \frac{1}{k}\sum_{i=0}^{k} y_i$$

ERM and Regularized Least Squares

1. Introduction to Regularization Networks

- Regularization Networks are a class of learning algorithms designed to prevent overfitting while maintaining generalization by incorporating regularization techniques.
- Focus on Tikhonov regularization, which balance empirical error with model complexity.

2. Empirical Risk Minimization (ERM):

- Empirical risk: $\hat{E}(f) = \frac{1}{n}\Sigma_{i=1}^{n} l(y_i, f(x_i))$ a proxy of uncomputable expected risk $E(f)$.
- ERM minimizes empirical risk over a hypothesis space H, but rich H can lead to overfitting.

3. Hypothesis Space and Linear Models:

- Hypothesis space: $H = \{f(x) = w^t x | x \in R^d\}$.
- Linear models define predictions using a weight vector w.

4. Tikhonov Regularization:

- Objective: $\min \hat{E}[f_\omega] + \lambda ||w||^2$
- $||w||^2$ is regularization term to stabilize solution. It avoids high variance by penalizing large weights this is crucial when dealing with noisy data.
- λ is regularization parameter. It controls the trade-off between empirical error and regularization term. High λ reduces overfitting but may cause underfitting.

5. Regularization least square:

- Loss function: $l(y, f(x)) = (y - w^t x)^2$.
- Optimization problem: $\min \frac{1}{n} ||Y_n - X_n w||^2 + \lambda ||w||^2$,
- Where X_n is the input matrix and Y_n is the output vector.
- The gradient of RLS loss function:

$$-\frac{2}{n} X_n^T (Y_n - X_n w)$$

6. Computational Aspects:

- Solution involves solving a linear system: $(X_n^T X_n + \lambda I)w = X_n^T Y_n$
- Time complexity: Training $O(nd^2)$ $(when\ n \gg d)$ Testing: $O(d)$.

7. Addressing Offset in Linear Models:

- The offset ensures model can represent linear functions that don't pass the origin.
- Including an offset b in the model $(f(x) = w^t x + b)$:
- Augment input space: $\bar{x} = (x, 1), \widetilde{\omega} = (w, b)$.
- Regularize only w not b to avoid bias towards zero offset.
- In regularization, we penalize the magnitude of weight vector to prevent overfitting. If w includes b as well, the regularization becomes $\Rightarrow ||\overline{\omega}||^2 = ||\omega||^2 + b^2$. b is also penalized here, and the function will be biased towards $b = 0$

8. Logistic Regression with Regularization:

- Logistic Loss function: $l(y, f_w(x)) = \log(1 + e^{-yw^t x})$.
- Optimization Via Gradient Descent: $w_{t+1} = w_t - \gamma \nabla(\hat{E}(f_w) + \lambda||w||^2)$
- The Gradient of Logistic loss function:

$$\nabla \hat{E}(f_w) = \frac{1}{n}\sum_{i=1}^{n} x_i \frac{-y_i}{1+e^{y_i x_i^T w_{t-1}}}$$

- Probabilistic Interpretation: $p(1|x) = \frac{e^{w^t x}}{1+ e^{w^t x}}$
- Logistic Regression is not just a classification problem. It also provides a probabilistic framework for predictions. So, the output of Logistic Regression can be interpreted as the probability of a data point belonging to a particular data.

9. Support Vector Machine (SVM):
   - Hinge Loss: $l(y, f_w(x)) = \max(0, 1 - yw^T x)$.

$$\mathrm{w}^* = \max_{\mathrm{w}\in\mathrm{R^d}} \min_{1\le i\le n} \mathrm{d}(\mathrm{x_i}, \mathrm{w})^2$$

$$w^* = \max_{w\in R^d} \min_{1\le i\le n} \frac{|x_i| - (x_i^T w)^2}{|w|^2}$$

   - The optimization problem: $\min||w||^2 + C\frac{1}{n}\sum_{i=0}^{n}\xi_i\, , subject\ to\ y_i w^T x_i \ge 1 - \xi_i$
   - SVM maximizes the margin between classes while allowing for some classification errors controls by C.

10. Maximum Margin Classifier
    - Maximize the margin distance between the separating hyperplane and the closest data points.
    - Margin: $m_i = y_i w^t x_i$
    - Select a hyperplane equidistance from the closest points of each class.

11. Dual Formulation of SVM
    - An alternative way to solve SVM optimization problem. Instead of directly optimizing the hyperplane, in the primal space, the dual formulation forces n Lagrange multipliers associated with the constraints
    - Solution is expressed as: $w = \sum_{i=0}^{n} \alpha_i y_i x_i$
    - Where α_i Lagrange multipliers are obtained from quadratic programming problem.
      - Sparsity: Only SVM has non-zero α
12. General Framework:
    - Regularized Empirical Risk Minimized (RERM): $min\frac{1}{n}\sum_{i=0}^{n} l(y_i w^T x_i) + \lambda||w||^2$

Sample Questions:

1. **What's Empirical Risk Minimization (ERM):**
   ERM is a framework for designing learning algorithms by minimizing empirical risk.

$$\hat{E}(f) = \frac{1}{n}\Sigma_{i=1}^{n} l(y_i, f(x_i))$$

   Where $l(y, f(x))$ is the loss function, n is the number of samples. It acts as a proxy for minimizing the expected risk $E(f)$, Which is not computable due to the unknow data distribution $p(x, y)$.

2. **Why is the expected risk $E(f)$ uncomputable?**
   The expected risk $E(f) = E[l(y, f(x)] = \int p(x, y) l(y, f(x)) dxdy$ depends on the true data distribution $p(x, y)$ which is typically unknown.

3. **What is the role of hypothesis space H in ERM?**
   The hypothesis space H is a set of functions f over which ERM minimizes the empirical risk.
   It should be:
   - Computationally feasible.
   - Be rich enough to capture underlying patterns in the data.

4. **What is an example of simple hypothesis space for linear models?**
   The hypothesis space for linear models is: $H = \{f(x) = w^t x | x \in R^d\}$.

5. **What issue arises with rich hypothesis spaces?**
   Rich hypothesis spaces can lead to overfitting, where the model fits the training data well but performs poorly on unseen data due to capturing noise or overly complex patterns.

6. **How do Tikhonov Regularization addresses overfitting?**
   Tikhonov Regularization adds a penalty term $\lambda||w||^2$ to objective function, controlling model complexity ensuring stability: $\min \hat{E}[f_\omega] + \lambda||w||^2$.

7. **What is the role of regularization parameter λ?**
   Regularization parameter λ balance the trade-off between fitting the data (minimizing empirical loss) and regularization (minimizing $||w||^2$).

8. **What is the regularizer $||w||^2$, and why it's important?**
   The regularizer $||w||^2$ penalizes large weights, promoting simpler models and preventing overfitting by limiting the complexity of learned function.

9. **What happens when $\lambda \to 0$ in Tikhonov Regularization?**
   The regularization term vanishes, and the solution minimizes empirical risk without any penalty, leading to potential overfitting.

10. **How does the choice of loss function affect the solution?**
    Different loss functions lead to different optimization problems and learning algorithms.
    - Squared loss -> Least Squares
    - Logistic loss -> Logistic Regression
    - Hinge loss -> SVM

11. **What is the object of Regularized Least Squares (RLS)?**
    RLS minimizes squared error with a regularization term: $min\frac{1}{n}\sum_{i=1}^{n}(y_i - w^t x_i)^2 + \lambda||w||^2$

12. **Write the RLS objective in matrix notation.**
    Using the input matrix X_n and output vector Y_n: $min\frac{1}{n}||Y_n - X_n w||^2 + \lambda||w||^2$

13. **How is the gradient of empirical risk computed for RLS?**
    The gradient is: $-\frac{2}{n}X_n^T(Y_n - X_n w)$

14. **What is the solution to the RLS optimization problem?**
    The solution satisfies: $w(X_n^T X_n + \lambda nI) = X_n^T Y_n$ where λ ensures matrix is invertible.

15. **Why is λ important for invertibility in RLS?**
    Adding λI ensures $X_n^T X_n + \lambda I$ is positive definitely, preventing issues when $X_n^T X_n$ is singular.

16. **Why is Cholesky decomposition is suitable for solving RLS?**
    Cholesky decomposition is efficient for symmetric positive definite matrices like $X_n^T X_n + \lambda I$ making it preferred method for solving a linear system in RLS.

17. **What is the computational complexity of RLS?**
    Training: $O(nd^2)$, *assuming* $n \gg d$, Testing $O(d)$.

18. **How does introducing offset affect linear models?**
    An offset b shifts the decision boundary, allowing the model to handle data that doesn't cross the centered at origin.
    The model becomes: $f(x) = w^T x + b$

19. **How is offset *b* incorporated in RLS?**
    By augmenting the input vector x with an additional dimension: $\bar{x} = (x, 1), \bar{\omega} = (\omega, b)$.

20. **Why should the offset not be penalized in regularization?**
    Penalizing the offset biases, the solution towards zero-offset models, which may not represent the data well.

21. **How is offset *b* is computed in centered data?**
    The offset is: $b^* = \bar{y} - \bar{x}^T w^*$ where $\bar{y}$ and $\bar{x}$ are the means of outputs and inputs, respectively.

22. **What is the advantage of centering the data in RLS?**
    Centering simplifies the computations by decoupling the offset b from the regularization term $||w||^2$

23. **How is the regularization problem modified to exclude *b*?**

$$min \frac{1}{n}\sum_{i=1}^{n}(y_i - w^t x_i - b)^2 + \lambda ||w||^2$$

24. **Why does centering data reduces the problem to standard RLS?**
    Centering transforms the data such that the offset b is absorbed into the mean subtraction, making the optimization problem equivalent to RLS without offset.

25. **What is the Tikhonov Regularization term for centered input?**
    For centered input, the regularization term remains $||w||^2$, unaffected by the offset.

26. **How does the RLS improves over standard linear regression?**
    The RLS adds a regularization term $\lambda||w||^2$, improving generalization and stability, while standard linear regression minimizes only the squared error.

27. **How does the choice of λ affect RLS solutions?**
    - Small λ: leads to low bias but high variance increasing the risk of overfitting.
    - Large λ: leads to high bias and low variance, risking the underfitting.

28. **What is the objective of Regularized Logistic Regression (RLR)?**
    RLR minimizes the regularized empirical risk with the logistic loss:

$$\min_{w \in \mathbb{R}^d} \hat{E}(f_w) + \lambda\|w\|^2, \hat{E}(f_w) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_w(x_i)),$$
$$\text{where } \ell(y, f_w(x)) = \log\left(1 + e^{-y f_w(x)}\right).$$

29. **Why is logistic loss function used in classification?**
    The logistic lost function: $\ell(y, f_w(x)) = \log\left(1 + e^{-y f_w(x)}\right)$, is differentiable, convex, and provides probabilistic outputs by modeling $p(y|x)$ with a logistic function.

30. **What is the probabilistic interpretation of Logistic Regression?**

The logistic regression model estimates the probability of $y = 1$ given x using logistic function $p(1|x) = \frac{e^{w^T x}}{1+e^{w^T x}}$

31. **How is gradient descent applied to RLR?**
    Gradient descent update weights w iteratively: $w_t = w_{t-1} - \gamma\nabla(\hat{E}(f_w) + \lambda\|w\|^2)$, where the gradient of logistic loss is : $\frac{\partial \hat{E}(f_w)}{\partial w} = \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i x_i}{1+e^{y_i w^T x_i}}$.

32. **What is the role of learning rate γ in gradient descent?**
    The learning rate γ controls the size of step in each iteration. A small γ leads to slow convergence, while a large γ risks the overshooting the optimal solution.

33. **How does the regularization improve Logistic Regression?**
    Regularization $\lambda||w||^2$ prevents overfitting by penalizing large weights, promoting generalization, especially when the dataset has noisy or high dimensional data.

34. **What is the difference between RLR and unregularized Logistic Regression?**
    RLR adds a regularization parameter $\lambda||w||^2$ to the optimization objective, which controls overfitting, whereas unregularized regression only minimizes logistic loss.

35. **What is the key property of Logistic function used in classification?**
    It maps real-valued inputs to probabilities in (0,1), making it suitable for binary classification.

36. **What is Support Vector Machine (SVM)?**
    SVMs are classification methods that finds the hyperplane that maximizes margin between two classes, ensuring robustness and good generalization.

37. **What is the Hinge loss function used in SVMs?**
    The hinge loss is: $l(y, f_w(x)) = \max(0, 1 - yw^T x)$ it penalizes predictions that either are incorrect or withing margin.

38. **Write the SVM objective with hinge loss.**
    The SVM optimization problem is: $min \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - yw^T x) + \lambda||w||^2$.

39. **What is margin in SVMs, and how is it defined?**
    The margin is the distance between hyperplane and the closest data points. For a data point (x_i, y_i), the margin is: $m_i = y_i w^T x_i$

40. **What is the geometric intuition behind the maximum margin classifier?**
    Classifies the training data perfectly, and maximizes the distance to the closest data points.

41. **How is the SVM margin maximized mathematically?**
    The margin is maximized by minimizing the $||w||^2$ under the constraint: $y_i w^T x_i \geq 1, \forall i \in \{1, \dots, n\}$.

42. **What is the primal form of the SVM optimization problem?**

$$\min_{w\in\mathbb{R}^d} ||w||^2 + C\frac{1}{n}\sum_{i=1}^{n} \xi_i$$
$$y_i w^T x_i \geq 1 - \xi_i, \xi_i \geq 0$$

43. **What are slack variables (ξ_i) in SVM?**

Slack variables allow some violation from the margin constraints, enabling SVM to handle non-linearly separable data. ξ_i measures the degree of violation for the i-th point.

44. **What does parameter C controls in SVM?**
The parameter C controls the trade-off between maximizing the margin and minimizing classification error:
- Large C: Focusing on minimizing error, leading to narrow margins.
- Small C: prioritizes larger margins, allowing more errors.

45. **What is the dual form of SVM optimization problem?**

$$\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j,$$

subject to:

$$0 \leq \alpha_i \leq C, \sum_{i=1}^{n} \alpha_i y_i = 0$$

46. **What are support vectors in SVM?**
Support vectors are data points associated with non-zero α_i in the dual solution. They lie on or within the margin and determine the hyperplane.

47. **How does the dual solution determine the SVM hyperplane?**
The hyperplane is defined as: $\sum_{i=1}^{n} \alpha_i y_i x_i$ where $\alpha_i > 0$ corresponds to support vector.

48. **Why is SVM robust to outliers when C is small?**
A small C allows larger margins and tolerate misclassifications, reducing the influence of outliers on the hyperplane.

49. **What is the computational advantage of the dual form in SVMs?**
The dual form operates in terms of α_i, requiring optimization over n variables (number of samples) instead of d variables (dimensionality of features).

50. **How's the kernel trick used in SVM?**
The kernel trick replaces dot product $x_i^T x_i$ with a kernel function $K(x_i x_i)$, enabling SVMs to learn non-linear decision boundaries in higher-dimensional feature space.

51. **What are common kernel functions in SVMs?**
- Linear kernel: $K(x_i x_i) = x_i^T x_i$
- Polynomial kernel: $K(x_i x_i) = (x_i^T x_i + c)^p$
- Gaussian kernel: $K(x_i x_i) = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2}\right)$

52. **Why does Hinge loss priorities large margins?**
The Hinge loss penalizes points within the margin or misclassified points more heavily, encouraging the model to maximize the margin.

53. **How does the quadratic programming formulation benefit SVMs?**
Quadratic programming ensures a unique global optimum for the SVM optimization problem, as the objective function is convex.

54. **What is the point-hyperplane distance in SVMs?**
The distance between a point x and the hyperplane $w^T x + b = 0$ is: $d(x, w) = \frac{||w^T x + b||}{||w||}$

55. How is the SVM margin maximized using constraints?
The margin is maximized by solving: $\min_{w,b} ||\omega||^2$ subject to: $y_i(w^T x_i + b) \geq 1, \forall i.$

56. What happens if data is not linearly separatable in SVMs?

The constraints are relaxed by slack variables ξ_i, and a penalty term is added to the objective function: $\min_{w,b,\xi} ||\omega||^2 + C\sum_{i=1}^{n} \xi_i$.

57. Why are SVMs are effective for high dimensional data?
SVMs maximize the margin, which reduces overfitting even in high-dimensional spaces. The use of kernels further allows efficient computation without explicitly increasing dimensionality.

Kernel functions and feature maps

1. **Extending to Non-Linear models:**
   - Linear models can be extended to non-linear problems by mapping input features into a higher-dimensional feature space using a feature map $\phi(\mathrm{x})$.
   - Linear model in the feature space: $f_w(x) = w^T\phi(\mathrm{x})$, where $\phi: R^d \to R^D$ transforms x into higher dimensional representation.

2. **Feature maps and non-linear representation:**
   - The feature map $\phi(\mathrm{x})$ allows the linear model to learn mode complex, non-linear patterns in the original input space.
   - Example: Mapping $x\ to\ \phi(\mathrm{x}) = (x^2, x, 1)$ enables the model to learn quadratic relationships.

3. **RLS Feature Space:**
   - Φ is the data matrix in feature space
   - $\mathrm{w} = (\Phi^T\Phi + \lambda nI)^{-1}\Phi^T y$
   - Direct computation in high dimensions space ($D \gg d$) is infeasible due to the size of matrices like $\Phi^T\Phi$ (of size $D \times \mathrm{D}$)
   - Efficient computation is achieved by the Representer Theorem, which reduces the solution space to depend only on the data points.

4. **Representer Theorem:**
   - Statement: for any regularized empirical risk minimization problem, the solution w^* can be expressed as: $w^* = \Phi^T c$ where $c \in R^n$ is a coefficient vector.
   - This reduces the optimization problem to computation involving $n \times n$ metrics instead of $\mathrm{D} \times \mathrm{D}$, making it computationally feasible.

5. **Kernel Trick:**
   - The kernel trick eliminates the need to explicitly compute $\Phi(\mathrm{x})$:
   - Kernel function: $K(x, x') = \phi(x)^T\phi(x')$ computes the inner product of feature space without explicitly mapping x,
   - Example of the function approximation: $f(x) = \sum_{i=1}^{n} c_i K(x, x_i)$

6. **Examples of Kernel functions:**
   - Linear Kernel: $K(x, x') = x^T x'$
   - Affine Kernel: $K(x, x') = x^T x' + \alpha^2$
   - Polynomial Kernel: $K(x, x') = (x^T x' + 1)^p$
   - Gaussian (RBF) Kernel: $K(x, x') = e^{-\frac{||x-x'||^2}{2\sigma^2}}$

7. **Properties of Kernel:**
   A function $K: R^d \times R^d \to R$ is a valid kernel if:
   - Symmetry: $K(x, x') = K(x', x)$
   - Positive Definiteness: for any $\{x_1, \ldots, x_n\} \subset R^d$, the matrix $K_{ij} = K(x, x')$ is positive definite.

8. **Computational Efficiency using kernels:**
   - Using kernels, the optimization problem depends only on the kernel matrix K:

$$f(x) = \sum_{i=1}^{n} c_i K(x, x_i)$$

9. Applications of Kernels:

- Kernel enables the use of linear in high-dimensional feature to solve non-linear problems.
- Classification: SVM
- Regression: KRLS

10. Advantages of Kernel Trick:

- Avoids explicit computation of $\phi(x)$, reducing the computational cost.
- Handle infinite-dimensional feature space, as in the case of Gaussian Kernel.

11. Feature map is a function: $\boldsymbol{\Phi}$: X → F

- Maps input data from the input space X into a new feature space F, where learning becomes tractable.
- Feature mapping allows linear models to approximate non-linear relationships.
- The choice of $\Phi(\mathrm{x})$ determines the complexity and expensiveness.
- By applying transformation $\Phi(\mathrm{x})$ the data becomes linearly separatable in higher dimension space.
- Feature maps allow nonlinear models to leverage linear methods, making computations straightforward.
- Though $\Phi(\mathrm{x})$ may lead to high-dimensional transformations, computation remains efficient because most methods depend only on inner products $(\Phi(\mathrm{x}), \Phi(x'))$

Sample Questions

1. **How does the feature map $\boldsymbol{\phi(x)}$ extend linear models to non-linear models?**
   The feature map $\phi(x)$ transforms input x from the original space R^d to a higher dimensional space R^D, enabling linear models represent non-linear patterns: $f_w(x) = w^T\phi(x)$.

2. **What is the computational challenge with high-dimensional feature maps?**
   In High-dimensional spaces ($D \gg d$) computing $\phi(x)$, $\Phi^T\Phi$ and directly solving for w becomes computationally expensive or infeasible due to the size of the matrices involved.

3. **What is Representer Theorem, and why is it important?**
   It states that solution w^* to a regularized empirical risk minimization problem can be expressed as $w^* = \Phi^T c$ where $C \in R^n$ is a coefficient vector. It reduces the problem to only depend on the data points, making computations feasible even in high dimensional spaces.

4. **How does the Representer Theorem simplify the computations in the feature space?**
   Instead of solving w in R^D, the problem is reduced to solving for c in R^n, this avoids the direct computations in high-dimensional feature space.

5. **Define a kernel function.**
   Kernel function$K(x, x')$ computes the inner products of feature vector in high-dimensional space without explicitly computing $\phi(x)$:
   $$K(x, x') = \phi(x)^T\,\phi(x')$$

6. **How does the kernel trick work?**
   Kernel trick replaces the explicit the computation of $\phi(x)$ with a kernel function $K(x, x')$, allowing efficient computation of inner products in the feature space.

7. **What is the mathematical expression for $\boldsymbol{f(x)}$ in terms of kernel?**
   $$f(x) = \sum_{i=1}^{n} c_i K(x, x_i)$$
   - Where c_i is the coefficient and $K(x, x_i)$ is the kernel function

8. **What is the advantage of kernel trick in machine learning models?**
The kernel trick enables learning non-linear patterns in the original input space without explicitly mapping data into high-dimensional feature space, reducing the computational cost.

9. **What are the properties of a valid kernel function?**
   - Symmetry: $K(x, x_i) = K(x_i, x)$
   - Positive definiteness: ensures the kernel can be used in optimization problems, such as solving linear systems.

10. **What are linear, affine, Polynomial, and RBF kernels**
    - Linear Kernel: corresponding the dot product of the input vector:
    $$K(x, x') = x^t x'$$
    - Affine Kernel: adds a constant term to the linear Kernel:
    $$K(x, x') = x^t x' + \alpha$$
    - Polynomial Kernel: enables the model to represent polynomial relationships of degree p
    $$K(x, x') = (x^t x' + 1)^p$$
    - RBF (Gaussian) Kernel: where the σ controls the kernel's bandwidth and determines the influence of points.
    $$K(x, x') = e^{\frac{-||x-x'||^2}{2\sigma^2}}$$

11. **Why is the Gaussian kernel effective for non-linear problems?**
The Gaussian kernel maps the inputs into an infinite-dimensional space, allowing the model to learn highly complex, non-linear patterns.

Neural Networks

1. **Definitions:**
   - Deep learning: A family of machine learning methods based on artificial neural networks (ANNs) with multiple layers for hierarchical feature extraction.
   - Artificial Neural Networks (ANNs): Computing systems inspired by biological neural networks, using interconnected nodes (neurons) for computations.
   - Neuron Firing: Outputs $y = 1$ if weighted sum $g(X) = \sum_{i=1}^{m} x_i$, exceeds the threshold θ, otherwise $y = 0$.
   - Activation Function:
   $$f\big(g(X)\big) = \begin{cases} 1 & if\ g(X) > \theta \\ 0 & other\ wise \end{cases}$$

2. **Single Layer Perceptron:**
   - Computes the weighted sum of input.
   - Applies activation function, typically a step function
   $$f(X) = \sigma(w_0 \sum_{i=1}^{m} w_i x_i) = \sigma(w_0 + X^T W)$$

3. **Activations Functions: Enable non-linear transformation, allowing neural networks approximate complex relationships.**
   - Sigmoid:
     - None-linear suitable for probabilistic interpretations.
     - Output layer of binary classification task
     - Gradient vanishing problem for very large or small data.

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

- Tanh:
    - It suffers from vanishing gradient problem
    - It has better gradient properties than sigmoid
    - It is scaled version of sigmoid

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} = 2\sigma(2x) - 1$$

- ReLU:
    - Very popular and simple. Its thresholds values below 0 (handles gradient vanishing)
    - It allows fast convergence of the optimization function (computationally efficient)

$$f(x) = \max(0, x)$$

- Leaky ReLU:
    - It aims to fix the dying ReLU problem

$$f(x) = \begin{cases} ax & if\ x < 0 \\ x & if\ x \geq 0 \end{cases}$$

- Softmax:
    - Softmax allows us to represent the probability distribution over M different classes
    - Best choice for multi-class classification

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{M} e^{z_k}}$$

4. **Multilayer Perceptron (MLP):**

- Extended single-layer perceptron by adding hidden layers.
- Each layer applies a linear transformation followed by a non-linear activation function.
- Architecture:
    - The dimensionality of the hidden layer determines the width
    - Number of layers determines the depth
    - Each hidden unit is associated with an activation function
    - $f_{layer,node}, w_{layer,node(to),input(from)}$

$$f_{1,n} = \sigma(\left(\sum_{i=1}^{2} w_{1,n,i}.x_i\right) + b_{1,n})$$

-
    - In an MLP, with inputs x_1, x_2, the number of unknown parameters is:

$$weights(6+9+3) + bias\ terms(7) = 25$$

5. **Model Capacity:**

- Refers model's ability to represent complex functions. It's determined by:
    - Architecture
    - The choice of activation functions
    - The number of trainable parameters
- A model with low capacity cannot capture the patterns of data leading to underfitting.
    - Low capacity: high bias, low variance
- A model with high capacity, can memorize the training data and lead to overfitting.
    - High capacity: low bias, high variance

6. **Feedforward Neural Networks**

- Input signals propagate layer by layer without cycles.
- Outputs depend only on the current input and model parameter

7. **Backpropagation and Training**

- Training involves adjusting the weights to minimize a loss function using gradient decent.

- Computes the gradients of loss with respect to weights using the chain rule.
- Efficiently propagate errors backwards through the network.

8. **Loss Function: measures the difference between the true value and predicted outputs.**
   - MSE for regression
   - Cross-entropy loss for classification

$$W^* = argmin \frac{1}{n}\sum_{i=1}^{N} l(F(X^i; W)y^i) = argmin\, J(W)$$
$$w_t = w_{t-1} + \gamma \nabla J(W_{t-1})$$

9. **The chain rule of derivation:**
   - The chain rule is the essence of DNN training, it allows to estimate gradient for high dimensional spaces from the partial derivatives with respect to each weight.

$$\frac{dF}{dx} = \frac{d}{dx}F(x) = \frac{d}{dx}f\left(g\left(h\left(u\left(v(x)\right)\right)\right)\right) = \frac{df}{dg}\cdot\frac{dg}{dh}\cdot\frac{dh}{du}\cdot\frac{du}{dv}\cdot\frac{dv}{dx}$$

10. **Bagging and dropout**
    - Bagging: is an ensemble learning technique aimed to reduce variance and improve the generalization of the model. It works by creating multiple subsets of dataset and combining their prediction.
      - bootstrap sampling: creating multiple training dataset by sampling from the original dataset. (each subset is used to train a different dataset)
      - train independent models on each sample.
      - prediction aggregation:
        - for classification : combine prediction using the majority voting.
        - for regression : using average.
      - This is computationally very expensive; dropout is a way to approximate the same behavior.
    - Dropout: At each step of the gradient descent some fraction of weights is dropped-out of each layer

11. **Gradient Descent:**
    - Batch Gradient Descent: Computes the gradient for the entire dataset on one iteration.
      - Convergence: Ensures Convergence to the global minimum for convex cost function
      - Pros:
        - Stable and consistent convergence
        - Global perspective: gathers knowledge from all the training samples reducing the risk of noisy updates
      - Cons:
        - Computationally expensive: Slow training time on large datasets
        - Memory intensive: storing and processing entire dataset can overwhelm the memory

$$W_{t+1} = W_t + \alpha\nabla_W J(W_t, X)$$
$$|W_{t+1} - Wt| < toll$$

    - Stochastic Gradient Descent: for each sample compute the gradient of cost function using single sample, updates the weights using the gradient, repeat the process for each sample in the dataset and several epochs.

- Pros:
    - Fast Update: each weight is updated after gradient of each sample is computed
    - Efficient memory Usage: requires to store one sample in memory in each iteration.
- Cons:
    - Noisy Convergence: The randomness in updates introduces noise, making it harder to converge to the exact minimum.

$$W_{t+1} = W_t + \alpha \nabla_W J(W_t, x_i)$$

- Mini-baches: Select a portion of training set of size s (mini-batch size) and update the weights after evaluating the loss function

$$W_{t+1} = W_t + \alpha \nabla_W J(W_t, x_{i:i+s})$$

Convolutional Neural Networks (CNN)

1. **Introduction to CNN:**
   - CNN are specialized type of NN that designed for data with grid-like topology, such as:
     - 1D grids: Time-series data
     - 2D grids: Images
   - They leverage two key parameters:
     - Sparse interaction: Reduced the number of connections compared to dense networks
     - Parameter sharing: Reuses the weights across different special locations.

2. **Sparse Interaction:**
   - Unlike dense networks where each output is connected to every input, CNNs restrict connections to a small **receptive field.** With benefits of:
   - Memory Efficiency: requires fewer parameters $O(k \times n)$ than dense networks $O(m \times n)$.
   - Statistical Efficiency: Improves generalization with fewer parameters.
   - Computational Efficiency: Reduces the complexity from $O(m \times n)$ to $O(k \times n)$ where k is kernel size.

3. **Parameter Sharing:**
   - In CNNs, the same filter (kernel) is applied across the input, detecting similar features in different regions.
   - This enables efficient feature detection and reduces the number of parameters.

4. **Convolution Operation:**
   - Convolution is a mathematical operation that involves sliding a filter(kernel) over the input to compute the feature map.
   - Formula for 2D convolution: $s(i,j) = \sum_m \sum_n I(i+m, j+n)k(m,n)$. Where I is the input, k is kernel, and $s(i,j)$ is output.

5. **Receptive Field:**
   - Each output in CNN corresponds to a specific receptive field in the input, determined by kernel size.

6. **Stride and Padding:**
   - Stride: determines the step of filter while sliding. Larger strides results in down sampling.
   - Padding: Adds zeros around the input to preserve the spatial dimensions after convolution.
   - Formula for output size: $O = \frac{W-K+2P}{S} + 1$ where W is input, K is kernel, P is padding, and S is stride.

7. **Pooling Layers:**
   - Pooling reduces special dimensions of the feature maps, improving the invariance to small shifts and reducing computational cost.
   - Max pooling: Retains the maximum value in the region.
   - Average pooling: Compute the average values of the region.

8. **Forward and Backpropagation in CNNs:**
   - During forward propagation, convolution and pooling are applied to transform inputs into feature maps.
   - In backpropagation:
     - Gradients are computed for each weight in the kernel by summing contributions from all positions in the output.

  - This enables updating weights to minimize the loss.

9. **Advantages of CNNs for Images:**
   - Reduced parameter count: Handle large inputs (high resolution images) efficiently.
   - Feature Hierarchy: Detect low-level features (edges, texture) in initial layers and high-level features (shapes, objects) in deeper layers.
   - Translation Invariance: Achieved through parameter sharing and pooling.

10. Typical CNN Architecture:
    - Convolution layer: Extract features using filters
    - Activation functions: Introduces non-linearity (ReLU).
    - Pooling layer: Reduces dimensionality and enhances variance.
    - Fully connected layer: Maps the extracted features to the output space.

11. From Feature Engineering to Feature Learning
    - CNN eliminates the need to manually feature engineering by learning hierarchical features directly from data.
12. Practical Applications:
    - CNNs are widely used in image classification, object detection, segmentation, and other computer vision tasks.

Bayesian Models

1. **Introduction to Bayesian Learning:**
   - Bayesian learning focuses on estimating the probability distribution of the data and parameters, contrasting by the frequentist methods that directly estimate the quantities of interest.
   - It uses prior knowledge to inform the learning process, updating the beliefs and the new data is observed.
   - Bayes' theorem:

$$P(w|D) = \frac{P(D|w)P(w)}{P(D)}$$

     - $P(w|D)$: Posterior (Update belief)
     - $P(D|w)$: Likelihood (how well the model explains the data)
     - $P(w)$: Prior (Initial belief about w)
     - $P(D)$: Evidence (normalization constant)

2. **Maximum Likelihood Estimation (MLE):**
   - MLE finds the parameters of the probabilistic model that maximize the likelihood of observed data.

$$\theta_{MLE} = \arg\max_{\theta} L\,(\theta|X)$$

   - Example of density estimation:
     - For data $x_1 \dots x_n$ assume to follow a Gaussian distribution, the likelihood is:

$$L(\mu) \propto e^{-\sum_{i=1}^{n} |x_i - \mu|^2}$$

- The MLE of the mean is:

$$\mu_n = \frac{1}{n}\sum_{i=1}^{n} x_i$$

   - Generalization: MLE can extend to multiple parameters (e.g., mean and variance) or multidimensional cases, such as multivariate Gaussian distributions.

3. **Comparison of MLE and MAP:**
   - MLE:
     - Considers only the likelihood.
     - No prior information is incorporated.
   - MAP:
     - Combines the likelihood with prior beliefs.
     - Incorporates regularization through the prior.

4. **Benefits of Bayesian learning**
   - Uncertainty Quantification: Bayesian methods not only provide parameter estimates but also uncertainty measures, such as variance and confidence intervals.
   - Flexibility: Can handle complex models and incorporate prior knowledge effectively.

5. **Limitation:**
   - Computational cost: Bayesian methods often require integration or sampling over distributions, which can be computationally expensive for high-dimensional problems.

Sample Questions

1. **What is the key difference between Maximum Likelihood Estimation (MLE) and Bayesian Learning?**
   - MLE estimates parameters by maximizing the likelihood function:
   $$\theta_{MLE} = \arg\max_{\theta} L\,(\theta|X)$$
   - Bayesian Learning estimates a posterior distribution over parameters using Baye's theorem:
   $$P(w|D) = \frac{P(D|w)P(w)}{P(D)}$$
   - MLE provides point estimate, Bayesian method incorporates prior knowledge and quantify uncertainty.

2. **What are the advantages and limitations of Bayesian Learning?**
   - **Pros:**
     - Uncertainty Quantification: Bayesian methods provide a distribution over parameters instead of a single estimate.
     - Regularization: Avoids overfitting by incorporating prior knowledge.
     - Flexibility: Can handle complex models and integrate domain-specific knowledge.
   - Cons:
     - Computational Cost: Bayesian inference often requires integration or approximation techniques
     - Choice of Prior: Incorrect priors can lead to biased results.
     - High-Dimensionality Issues: Exact Bayesian inference becomes infeasible for large-scale problems

3. **Why is MLE sensitive to overfitting?**
   - MLE only maximizes the likelihood, which can fit noise in small datasets.
   - No regularization is applied, leading to high variance in small sample sizes.

4. **What is the role of the likelihood function in Bayesian inference?**
   - The likelihood function measures how well a given parameter explains the observed data.
   - In Bayesian inference, the likelihood is combined with the prior to compute the posterior.

Decision Tree

1. Partition-Based Estimators

- Partition-based methods divide the input space X into disjoint sets (cells). Each cell has its own local estimate.
- Dyadic Partition
    - The input space X is divided into cubes of side length 2^{-k}
    - This ensures a hierarchical structure of partitions.

2. Piecewise Constant Estimators

- In each cell, the function is approximated by a constant.
- The goal is to minimize:

$$\min_{c_1,\ldots,c_J \in R} \sum_{j=1}^{J} \sum_{x_i \in A_j} (y_i - c_j)^2$$

- The optimal solution for each cell is:

$$\hat{c}_j = \frac{1}{n_j} \sum_{x_i \in A_j} y_i$$

    - Where n_j is the number of points in each cell
- This is related to ERM, where the hypothesis space consists of piecewise constant functions.

3. Piecewise Linear Estimators

- Instead of a constant, a linear function is fit within each cell:

$$f(x) = \sum_{j=1}^{J} (w_j^T x + b_j) 1_{A_j}(x)$$

- The estimation problem reduces to Ordinary Least Squares (OLS) in each partition.

4. General Partition-Based Estimators

- The function in each partition can belong to any hypothesis class G (e.g., neural networks, reproducing kernel Hilbert spaces (RKHS), etc.).
- ERM is performed separately in each partition.

5. Partition Trees and Decision Trees

- Partition trees generalize the previous ideas by creating hierarchical partitions.
- Partition Tree Estimators
    - Each level q of the partition has J_q cells.
    - The function space is:

$$H_{G,q} = \{f : X \to R \mid f(x) = \sum_{j=1}^{J_q} g(x) 1_{A_j}(x), g \in G\}$$

- The estimation problem remains an ERM problem within each partition.

6. Uniform vs. Adaptive Partitioning

- Uniform partitioning: The scale parameter q acts as a regularization parameter.
- Adaptive partitioning: The cells are refined dynamically based on the loss function:

$$L_A = \min_{f \in H} \frac{1}{n_A} \sum_{x_i \in A} \left(y_i - f_A(x_i)\right)^2$$

- A cell is kept if $L_A \leq \tau$, otherwise, it is split.
- The threshold τ acts as a regularization parameter.

7. Decision Tree

- Decision trees use a greedy approach to partition the space.
- The partitioning criterion minimizes:

$$\min_{j,s}\left[L_{A_1(s,j)} + L_{A_2(s,j)}\right]$$

- Recursive splitting is applied to further refine the partitions.
- Alternative error measures like cross-entropy (for classification) or Gini index are used.

8. Kernel Regression

- Kernel regression shifts the focus from discrete partitions to a continuous weighting approach.

9. Nearest-Neighbor-Like Estimation

- Instead of fixed cells, each point is assigned a local weight:

$$f\widehat{(x)} = \arg\min_{c \in R} \sum_{i=1}^{n} (y_i - c)^2 1_{(|x - x_i| \leq \tau)}$$

- This is equivalent to nearest neighbor regression, where only the closest points contribute to the estimate.

10. Nadaraya-Watson Kernel Regression

- Instead of a hard threshold, a kernel function assigns weights to nearby points:

$$f\widehat{(x)} = \frac{\sum_{i=1}^{n} y_i k(x_i, x)}{\sum_{i=1}^{n} k(x_i, x)}$$

- $k(x_i, x)$ is a kernel function that determines how much weight is given to x_i when estimating $f(x)$.
- This is a weighted averaging method where closer points have higher influence.

11. Locally Linear Kernel Regression

- Instead of estimating a constant, a linear model is fitted locally:

$$f(x) = \hat{w}^T x + \hat{b}$$

- The weights are computed by solving:

$$\left(\hat{w}, \hat{b}\right) = \arg \min_{(w,b) \in R^{d+1}} \sum_{i=1}^{n} (y_i - w^T(x_i - x) - b)^2 k(x_i, x)$$

- This improves accuracy by allowing for local variations in slope.

12. Key Properties of Kernel Regression

- No optimization required: The solution is computed directly via weighted sums.
- Nonparametric: The function estimate adapts to the data, avoiding strong assumptions.
- Locally Adaptive: Can capture local structures in the data but does not impose global smoothness.

Sample Questions

1. How do decision trees lead to overfitting, and how can it be prevented?
   - Deep trees capture noise instead of general patterns, reducing generalization ability.
   - Prevention: Pruning, Max Depth, Random Forest

2. How does Locally Weighted Regression differ from Ordinary Least Squares (OLS)?
    - OLS: Finds one global linear model for the data.
    - Locally Weighted Regression: Fits a linear model locally using a kernel-weighted loss function:

$$(\hat{w}, \hat{b}) = \arg \min_{(w,b)\in R^{d+1}} \sum_{i=1}^{n} (y_i - w^T(x_i - x) - b)^2 k(x_i, x)$$

        - It can model non-linearity better than OLS.

3. How does adaptive partitioning work in decision trees?
    - The partitioning splits regions dynamically based on data distribution.
    - A split is performed if the local loss function exceeds a threshold:

$$L_A = \min_{f\in H} \frac{1}{n_A} \sum_{x_i\in A} \left(y_i - f_A(x_i)\right)^2$$

    - Effect: More splits in complex areas, fewer in smooth regions.

Variable Selection

1. Introduction:
- In machine learning, prediction alone is not enough; interpretability is also crucial.
- Interpretability depends on detecting key variables that contribute to predictions.
- The goal is to select relevant variables to enhance prediction accuracy while reducing complexity.

2. Linear Models and Sparsity
- The model is defined as:

$$f_w(x) = w^T x = \sum_{i=1}^{v} w_j x_j$$

    - x_j represents the input features (pixels, word count, …)
    - Variable selection helps identify features that significantly impact prediction
    - Sparsity assumption: The best model is often sparse, meaning only a few weights w_j are nonzero.

4. High-Dimensional Statistics
- Traditional models assume $n \gg D$ (more samples than features) which result in overdetermined system.
- In modern settings, $n \ll D$ (more features than samples) which result in underdetermined system.
- Buzzwords: compressed sensing, high-dimensional statistics

5. Brute Force Approach (Hard)
- Selecting features using combinatorial search (checking all subsets of features) is computationally infeasible

$$\min_{w\in R^D} \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f_w(x_i)\right)^2 + \lambda|w|_0$$

   - The l_0 norm counts non-zero coefficients.

$$l_0 \to ||w||_0 = \{j|w^j \neq 0\}$$

   - Computational complexity grows exponentially.
- Solution: Approximation Approaches
   - Greedy methods (Matching Pursuit, Orthogonal Matching Pursuit)
   - Convex Relation (LASSO, Elastic Net)

6. Greedy Methods: Matching Pursuit (MP)

- Iteratively select features that are most correlated with the residual error.
- Steps:
   - Initialize: Residual $r_0 = Y$, weight vector $w_0 = [\dots 0 \dots]$, and selected feature $I_0 = \emptyset$
   - Select the variable that most correlated with the residual.

$$k = \arg\max_{j=1,\dots,D} a_j\,, \quad a_j = \frac{(r_{i-1}^T X^j)^2}{|X^j|^2},$$

   - Update the coefficient vector w.

$$w_t = \left(\widehat{X_t^T X_t}\right)^{-1} \widehat{X_t Y}$$

   - Update the residual.

$$r_t = \hat{Y} - \hat{X} w_t$$

7. Orthogonal Matching Pursuit (OMP)

- Instead of simple updates, OMP solves a least squares problem:

$$w_i = \arg\min_{w\in R^D} |\,Y_n - X_n M_I w|^2$$

   - Provides a more accurate approximation
   - Works only if true solution is sparse and the features are not too corelated

8. Convex Relaxation: LASSO

- Instead of $l_0 - norm$, uses $l_1 - norm$:

$$|w|_1 = \sum_{j=1}^{D}|w^j|$$

   - It's the sum of absolute values of weights
- The optimization problem becomes:

$$\min_{w\in R^D} \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f_w(x_i)\right)^2 + \lambda|w|_1$$

- Convex can be optimized efficiently.
- Encourages spare solutions (some weights are exactly zero).

9. Iterative Soft Thresholding Algorithm (ISTA)

- A gradient-based approach to solve LASSO:

$$w_i = S_{\lambda\gamma}\left(w_{i-1} - \frac{2\gamma}{n} X_n^T (Y_n - X_n w_{i-1})\right)$$

- Soft thresholding function ensures sparsity:

$$S_\alpha(u) = \text{sign}(u)\cdot \max(|u| - \alpha, 0)$$

- Converges under proper step-size selection:

$$\gamma = \frac{n}{2|X_n^T X_n|}$$

10. Elastic Net: Combining LASSO and Ridge

- Problem with LASSO: When features are highly correlated, it randomly selects one.
- Solution: Elastic Net combines LASSO and Ridge regression:

$$\min_{w \in R^D} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f_w(x_i)\right)^2 + \lambda(\alpha|w|_1 + (1-\alpha)|w|_2^2)$$

  - $l_1 - norm$ Sparsity
  - $l_2 - norm$ Stability

11. ISTA for Elastic Net

- Modified iterative update:

$$w_0 = 0 \ , w_i = S_{\lambda\alpha\gamma}\left(\left(1 - \lambda\gamma(1-\alpha)\right)w_{i-1} - \frac{2\gamma}{n} X_n^T (Y_n - X_n w_{i-1})\right), \ \ i = 1, \dots, T_{max}$$

  - At each iteration a non-linear soft thresholding operator is applied to the gradient step.
- The iteration should be run until a convergence criterion is met or a maximum number of iteration T_{max} is reached.
- To ensure convergence we should choose a step-size:

$$\gamma = \frac{n}{2\left(|X_n^T X_n| + \lambda(1-\alpha)\right)}$$

12. Conclusion:

- Feature selection improves interpretability and reduces complexity
- Brute force search is infeasible, approximation methods are needed.
  - Greedy methods: Matching Pursuit (MP), Orthogonal Matching Pursuit (OMP)
  - Convex Relation (LASSO, Elastic Net)
- LASSO uses l_1-norm for sparsity.
- Elastic Net blends with LASSO and Ridge for better stability.
- ISTA provides an efficient way to solve LASSO and Elastic Net.

Sample Questions

1. **What's the key assumption behind sparsity-based models in machine learning?**
The key assumption is that the best predictive model only relies on small subset of features. This means that most features weights in the model should be zero, leading to a sparse solution.

2. **Explain why the l_0-norm regularization is computationally infeasible.**
l_0-norm regularization requires a combinatorial search over all possible features subsets to determine which subset minimizes the loss. This result is an NP-hard problem, making it infeasible for large datasets.

3. **Why l_1-norm regularization (LASSO) preferred over l_0-norm in practice?**
l_1-norm regularization provides a convex relaxation of the l_0-norm making the optimization problem computationally tractable. It encourages sparsity while allowing efficient solutions using gradient-based methods.

4. **How does the iterative soft thresholding algorithm (ISTA) work for LASSO?**
ISTA iteratively updates the weight vector using the following update rule:

$$w_i = S_{\lambda\alpha\gamma}\left(w_{i-1} - \frac{2\gamma}{n} X_n^T (Y_n - X_n w_{i-1})\right)$$

Where $S_{\lambda\alpha\gamma}$ is the soft-threshold operator that shrinks some weights to zero.

5. **Compare Matching Pursuit (MP) and Orthogonal Matching Pursuit (OMP).**
   - MP: Greedily selects the features that correlates the most with the residual and updates weights sequentially.

- OMP: Instead of updating the weight sequentially, it orthogonalizes the selected features to improve accuracy.

6. **When should Elastic Net regularization be used instead of LASSO?**
   Elastic Net should be used when the features are highly correlated, as LASSO tends to select one feature arbitrarily. Elastic Net balances l_1-norm sparsity with l_2-norm stability, preventing over selection of a single features.

7. **Given the optimization problem for Elastic Net, what does the hyperparameter α control?**

$$\min_{w \in R^D} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f_w(x_i)\right)^2 + \lambda(\alpha |w|_1 + (1-\alpha)|w|_2^2)$$

   - $\alpha = 1$: LASSO behavior (sparse solution)
   - $\alpha = 0$: Ridge Regression (No sparsity, only regularization)
   - $0 < \alpha < 1$ A balance between sparsity and stability

8. What is the main difference between Tikhonov regularization (Ridge) and LASSO?
   - Ridge regression (l_2-norm) shrinks all weights but doesn't enforce sparsity (no feature selection).
   - LASSO (l_1-norm) forces some weights to be exactly zero, leading to feature selection

9. **What are the key steps in the Matching Pursuit (MP) algorithm?**
   - Initialize residual $r_0 = Y_n$, coefficient vector $w = 0$.
   - Select the variable the correlates the most with the residual.
   - Update the coefficient vector for selected variable.
   - Update the residual.
   - Repeat until a stopping criterion is met.

10. **What is the main limitation of LASSO when handling highly correlated features?**
    LASSO randomly selects one feature from a group of correlated features while ignoring the rest, potentially losing important information. Elastic Net solves this by incorporating both l_1 and l_2 penalties.