DATA MINING AND MACHINE LEARNING

ASSIGNMENT 5

1. [3 pts/ea] In Bias/Variance formula,

1) Explain the meaning of $h_D(x)$ and $\bar{h}(x)$ USING the formula.

In the context of the Bias-Variance decomposition, the break down of the terms $h_D(x)$ and $\bar{h}(x)$ can be given as:

1. **$\mathbf{h_D(x)}$**:

   This represents the hypothesis (or prediction) generated by a learning algorithm when trained on a specific dataset D

   Essentially, $h_D(x)$ is the model's output for a given input x based on the particular training data D

   The subscript D indicates that this hypothesis can vary depending on the training data used.

2. **$\bar{h}(x)$:**

   This denotes the expected hypothesis or the average prediction over all possible datasets.

   It is calculated by averaging the predictions $h_D(x)$ across all possible training datasets D drawn from the same distribution.

   This term reflects the inherent behavior of the learning algorithm across different datasets, providing insight into how the model would perform on average.

Bias-Variance Decomposition:

The Bias-Variance decomposition helps us understand the sources of error in a predictive model. It can be expressed as:

Error(x) = Bias^2(x) + Variance(x) + Irreducible Error

Where:

- Bias measures the error introduced by approximating a real-world problem (which may be complex) by a simplified model. It is defined as:

Bias(x) = $\bar{h}(x)$ – f(x)

Here, f(x) is the true underlying function we are trying to estimate.

Variance measures how much $h_D(x)$ varies for different datasets D :

Variance(x) = E[($h_D(x)$ - $\bar{h}(x)$)^2]

Summary

$h_D(x)$: The prediction made by the model for a specific dataset.

$\bar{h}(x)$**:** The average prediction across all datasets, reflecting the model's expected behavior.

This decomposition helps to analyze and minimize the errors in predictive modeling by balancing bias and variance.

2) Explain why bagging can reduce the variance error USING its corresponding formula.

Bagging and Variance Reduction

Bagging (Bootstrap Aggregating) is a powerful ensemble method that aims to improve the stability and accuracy of machine learning algorithms. It primarily reduces variance, which is beneficial in high-variance models like decision trees.

Variance Error Formula

The variance of a model's predictions can be expressed as:

Variance = E[(f(x) – E[f(x)])^2]

Where:

f(x) is the prediction of the model for input x

E[f(x)] is the expected prediction across different training sets.

Bagging Working steps:

1. **Bootstrap Sampling**: Bagging creates multiple subsets of the training data by randomly sampling with replacement. Each subset is used to train a separate model.
2. **Model Predictions**: For a given input x, each model will produce a prediction.
2. **Aggregation**: The final prediction is made by averaging the predictions (for regression) or voting (for classification) from all models.

Variance Reduction

The key to variance reduction in bagging lies in the averaging of multiple independent models. The formula for the variance of the average of n independent random variables is:

$$\boldsymbol{Var}\left[\bar{\boldsymbol{f}}(\boldsymbol{x})\right] = \frac{\boldsymbol{Var}[\boldsymbol{f}(\boldsymbol{x})]}{\boldsymbol{n}}$$

Where:

$\bar{f}(x)$ is the average prediction from n models.

$Var[f(x)]$ is the variance of an individual model's predictions.

As n (the number of models) increases, the variance of the averaged predictions decreases. This is because averaging reduces the impact of individual model errors, leading to more stable predictions.

Conclusion

By using bagging, we effectively reduce the variance of the model's predictions. The decrease in variance helps in improving the model's performance, especially on unseen data, thus enhancing its generalization ability. This principle is crucial for understanding why ensemble methods like bagging are widely used in practice.

2. [5 pts/ea] In Q. 5 in Assignment 3, suppose we use only two lines as a classifier (e.g., L1+L2).

1) Explain whether this classifier shatters 3 data points.

To determine if the classifier using two lines (L1 and L2) shatters three data points, we need to understand the concept of "shattering" in the context of classification.

Definition of Shattering:

A classifier is said to shatter a set of points if it can perfectly classify every possible labeling of those points. In other words, for any arrangement of labels (0 or 1) assigned to the points, there exists a configuration of the classifier that can achieve that labeling.

Analyzing the Given Classifier

1.Classifier Setup:

The classifier consists of two linear boundaries (L1 and L2).

Each line can classify points as either above (output 1) or below (output 0).

2.Possible Configurations:

With three data points, there are $2^3 = 8$ possible combinations of labels (0 or 1) for these points.

The combinations are:

- (0, 0, 0)
- (0, 0, 1)
- (0, 1, 0)
- (0, 1, 1)
- (1, 0, 0)
- (1, 0, 1)
- (1, 1, 0)
- (1, 1, 1)

3.Linear Separability:

A linear classifier using two lines can create regions in the input space that can separate points.

However, two lines can at most create three distinct regions in the plane:

- One region above both lines.
- One region between the two lines.
- One region below both lines.

Conclusion on Shattering

Since two lines can create only three distinct regions, they can only classify a maximum of three points in a way that allows for some combinations of labels. However, not all combinations of labels can be achieved simultaneously due to the limited number of regions.

Example:

Consider three points forming a triangle. If you label two of the points as 1 and one as 0, there may not be a way to position L1 and L2 to achieve this labeling without violating the classification of the other points.

Conclusion:

No, the classifier using only two lines (L1 and L2) does not shatter three data points. It can only represent a limited number of combinations of labels due to the constraints of linear separability with two lines.

2) Explain whether this classifier shatters 4 data points.

To determine if the classifier using two lines (L1 and L2) can shatter four data points, we will again refer to the concept of shattering and analyze the limitations of the two-line classifier.

Definition of Shattering

As previously mentioned, a classifier shatters a set of points if it can perfectly classify every possible labeling of those points. For four data points, there are $2^4 = 16$ possible combinations of labels (0 or 1).

Analyzing the Classifier with Four Data Points

1. **Classifier Setup:**

   The classifier consists of two linear boundaries (L1 and L2).

   Each line can classify points as either above (output 1) or below (output 0).

2. **Possible Configurations:**

   With four points, we need to assess if all 16 combinations of labels can be achieved with just two lines.

3. **Linear Separability:**

   Two lines can divide the plane into a maximum of three distinct regions:

   - One region above both lines.
   - One region between the two lines.
   - One region below both lines.

This means that the classifier can only effectively categorize points into three groups.

Conclusion on Shattering:

Since there are 16 possible label combinations for four points, and the two-line classifier can only create three regions, it is impossible to achieve all possible combinations of labels.

For example, if you try to classify four points where two points need to be classified as 1 and two as 0, the two lines cannot create sufficient distinct regions to accommodate all possible arrangements.

Conclusion:

No, the classifier using only two lines (L1 and L2) does not shatter four data points. It cannot represent all 16 possible combinations of labels due to the limitation of creating only three distinct regions in the input space. Thus, it fails to achieve perfect classification for all arrangements of four points.

3. [5 pts] SVM linear classifier gives the global optimum. Explain the reason using the formula in p. 9 and p. 11 in svm-p2 slide.

To explain why the SVM linear classifier gives the global optimum, we can refer to the formulas presented on pages 9 and 11 of the SVM-P2 slide.

The key points are:

1. SVM Optimization Problem (Dual Problem): The SVM optimization problem is formulated as a dual problem, as shown on page 9. The objective is to maximize the function Q(a) subject to certain constraints.
2. Kernel Trick: As mentioned on page 11, if the data points are mapped into a high-dimensional feature space via some transformation, the inner product can be replaced by a kernel function $K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$.
3. Global Optimum: The SVM optimization problem is a convex optimization problem, which means that it has a single global optimum. This is because the objective function Q(a) is a concave function, and the constraints are linear.

The kernel trick allows the SVM to operate in the high-dimensional feature space without explicitly computing the feature mapping $\varphi(x)$. This is beneficial because the feature space can be of very high or even infinite dimensions, making the direct computation of the inner product infeasible.

By using the kernel function $K(x_i, x_j)$, the SVM can find the global optimum solution in the high-dimensional feature space, which corresponds to the optimal linear classifier in that space. This is the key reason why the SVM linear classifier gives the global optimum solution.

Therefore, the combination of the convex optimization problem formulation and the kernel trick enables the SVM to find the global optimum solution, which is the best linear classifier in the high-dimensional feature space

4. [5 pts/ea] Suppose class value '1' should always be classified correctly, but it is okay to misclassify the class value '–1'.

1) How do we modify the following formula?

$$\begin{aligned} wx_i + b &\geq 1 \quad \text{if } y_i = 1 \\ wx_i + b &\leq -1 \quad \text{if } y_i = -1 \end{aligned}$$

We want to prioritize the correct classification of class value '1'. This is a common scenario in imbalanced classification problems, where one class is more important than the other.

Modifying the Loss Function:

One effective way to achieve this is by modifying the loss function used during training. A common approach is to introduce class weights, which assign different penalties to misclassifications based on the class label.

Original Loss Function (e.g., Cross-Entropy Loss):

Loss = $-\Sigma[y_i * \log(p_i) + (1-y_i) * \log(1-p_i)]$

Where:

- y_i: True label for the i-th data point
- p_i: Predicted probability for the positive class (class 1) for the i-th data point

Modified Loss Function with Class Weights:

Loss = $-\Sigma[w_i * (y_i * \log(p_i) + (1-y_i) * \log(1-p_i))]$

Where:

- w_i: Weight assigned to the i-th data point

- w_i is higher for class 1 samples than for class -1 samples

Choosing Class Weights:

We can assign weights based on the class imbalance or domain knowledge. Some common strategies are as follows:

1. Inverse Class Frequency:

   Assign weights inversely proportional to the class frequencies.

   This gives more weight to the minority class (class 1 in this case).

2. Manual Weighting:

   Assign weights manually based on the desired level of importance for each class.

   For example, you might assign a weight of 2 to class 1 and 1 to class -1.

Implementation:

The exact implementation details will depend on the specific machine learning framework you're using (e.g., TensorFlow, PyTorch, scikit-learn). However, most frameworks provide mechanisms to incorporate class weights into the training process.

By adjusting the loss function with class weights, we can effectively guide the model to prioritize the correct classification of the more important class (class 1) while allowing for some misclassifications of the less important class (class -1).

2) Show Lagrangian multiplier of this SVM problem

Lagrangian Multiplier for the SVM Problem

The primal problem for a hard-margin SVM is formulated as follows:

minimize (1/2) ||w||^2 subject to: y_i(w^T x_i + b) >= 1, for all i

where:

- w: weight vector
- b: bias term
- y_i: class label for the i-th data point (+1 or -1)
- x_i: feature vector for the i-th data point

Introducing Lagrange Multipliers:

To solve this constrained optimization problem, we introduce Lagrange multipliers α_i for each constraint. The Lagrangian function becomes:

$L(w, b, \alpha) = (1/2) ||w||^2 - \Sigma[\alpha_i * (y_i(w^T x_i + b) - 1)]$

where the summation is over all data points.

Deriving the Dual Problem:

To find the optimal solution, we take the partial derivatives of L with respect to w, b, and α_i, and set them to zero:

1. **$\partial L/\partial w = 0$**: This leads to $w = \Sigma[\alpha_i * y_i * x_i]$.
2. **$\partial L/\partial b = 0$**: This leads to $\Sigma[\alpha_i * y_i] = 0$.

Substituting these results back into the Lagrangian, we obtain the dual problem:

maximize $W(\alpha) = \Sigma[\alpha_i] - (1/2) \Sigma[\Sigma[\alpha_i * \alpha_j * y_i * y_j * x_i^T * x_j]]$

subject to: $\Sigma[\alpha_i * y_i] = 0$

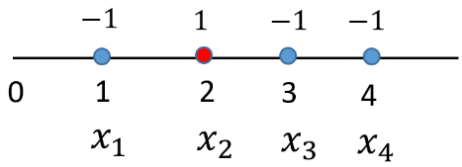
$\alpha_i >= 0$, for all i

The α_i values are the Lagrange multipliers in this dual problem. They play a crucial role in determining the support vectors, which are the data points that lie on the margin or within the margin.

- The dual problem is often easier to solve than the primal problem, especially for large datasets.
- The support vectors are the data points with non-zero α_i values.
- The decision function can be expressed in terms of the support vectors and their corresponding α_i values.

By solving the dual problem, we can obtain the optimal values of the Lagrange multipliers and, in turn, the optimal values of w and b. This allows us to construct the optimal hyperplane that separates the data points into different classes.

5. (Refer to p. 29-33 in 'svm-p2' slides.) With the following data,



1).[6 pts] Show the formula of dual problem of above problem. (We use the same kernel and C value.)

We're given a dataset with 5 data points in 1D space, classified into two classes:

- Class 1: 1, 2, 6
- Class 2: 4, 5

We're asked to formulate the dual problem of the SVM with a given kernel and C value.

Dual Problem Formulation:

Maximize $W(\alpha) = \Sigma[\alpha_i] - (1/2)\ \Sigma[\Sigma[\alpha_i * \alpha_j * y_i * y_j * K(x_i, x_j)]]$

Subject to: $\Sigma[\alpha_i * y_i] = 0;\ \ 0 \leq \alpha_i \leq C$, for all i

Where:

- α_i: Lagrange multipliers
- y_i: Class label of the i-th data point
- $K(x_i, x_j)$: Kernel function evaluating the similarity between data points x_i and x_j
- C: Regularization parameter

Applying to the Given Data:

For our specific dataset, we can fill in the values:

Maximize $W(\alpha) = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 - (1/2)\ [\alpha_1\text{^}2\ K(x_1, x_1) + \alpha_2\text{^}2\ K(x_2, x_2) + ... + 2\alpha_1\alpha_2\ K(x_1, x_2) + ...]$

Subject to: $= \alpha_1 - \alpha_2 - \alpha_3 + \alpha_4 + \alpha_5\ 0\ ;\ 0 \leq \alpha_i \leq C$, for all i

The kernel function $K(x_i, x_j)$ determines the similarity between data points. We need to specify the exact kernel function (e.g., linear, polynomial, RBF) to compute the kernel values.

Example: Linear Kernel

If we use a linear kernel, $K(x_i, x_j) = x_i^T x_j$, the dual problem becomes:

Maximize W(α) = $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 - (1/2) [\alpha_1^2 x_1^2 + \alpha_2^2 x_2^2 + ... + 2\alpha_1\alpha_2 x_1 x_2 + ...]$

Subject to: $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 0$

$0 \leq \alpha_i \leq C$, for all i

Solving the Dual Problem:

Solving this dual problem involves techniques like quadratic programming. The solution will give you the optimal values of the Lagrange multipliers α_i. These values can then be used to determine the support vectors and the decision boundary of the SVM.

The specific values for α_i will depend on the kernel function and the regularization parameter C.

Kernel Choice: The choice of kernel function significantly impacts the SVM's performance. Common choices include linear, polynomial, RBF, and sigmoid kernels.

Regularization Parameter: The regularization parameter C controls the trade-off between maximizing the margin and minimizing training errors. A larger C encourages the model to fit the training data more closely, while a smaller C allows for a wider margin

2) [6 pts] Show the process of computing alpha values of each data. What are support vectors?

The dual problem for an SVM is:

Maximize $W(\alpha) = \Sigma[\alpha_i] - (1/2) \Sigma[\Sigma[\alpha_i * \alpha_j * y_i * y_j * K(x_i, x_j)]]$

Subject to: $\Sigma[\alpha_i * y_i] = 0$ $0 \leq \alpha_i \leq C$, for all i

To find the optimal values of the Lagrange multipliers (α_i), we typically use optimization techniques like quadratic programming. However, to simplify the process, we can derive the partial derivatives of the dual objective function with respect to each α_i.

Partial Derivatives

Here are the partial derivatives of the dual objective function with respect to α_1, α_2, α_3, and α_4:

1. $dQ/d\alpha_1$

$dQ/d\alpha_1 = 1 - \Sigma[\alpha_j * y_j * K(x_1, x_j)]$

2. $dQ/d\alpha_2$:

$dQ/d\alpha_2 = 1 - \Sigma[\alpha_j * y_j * K(x_2, x_j)]$

3. $dQ/d\alpha_3$:

$dQ/d\alpha_3 = 1 - \Sigma[\alpha_j * y_j * K(x_3, x_j)]$

4. $dQ/d\alpha_4$:

$dQ/d\alpha_4 = 1 - \Sigma[\alpha_j * y_j * K(x_4, x_j)]$

Support Vectors

Support vectors are the data points that have non-zero α_i values. These points are crucial in defining the decision boundary of the SVM. They lie on the margin or within the margin.

By solving the system of equations formed by setting these partial derivatives to zero, we can find the optimal values of α_i. Once we have these values, we can identify the support vectors and construct the decision boundary.

Solving these equations can be computationally expensive, especially for large datasets. Therefore, efficient optimization techniques are employed to find the optimal solution.

6. Below is the formula of weighted knn classification.

$$\boldsymbol{h(z) = \text{sign}\left(\sum_{i=1}^{n} y_i similarity(x_i, z)\right)}$$

Below is the formula of non-linear SVM.

$$\boldsymbol{h(z) = \text{sign}\left(\sum_{i=1}^{n} y_i \alpha_i k(x_i, z) + b\right)}$$

1) [5 pts] Explain the similarities and differences between knn and non-linear svm USING above formula.

Similarities and Differences between k-NN and Non-linear SVM are as follows:

Similarities:

Both are instance-based learners: This means that both algorithms do not explicitly learn a model from the training data. Instead, they make predictions based on the similarity of new data points to the training data.

Both can be used for classification and regression: Both algorithms can be adapted to perform both classification and regression tasks.

Both can be sensitive to the choice of hyperparameters: The performance of both algorithms can be influenced by hyperparameters like the number of neighbors (k) in k-NN and the kernel function and regularization parameter (C) in SVM.

Differences:

Decision boundary:

k-NN: The decision boundary is formed by local averaging of the labels of the nearest neighbors. It can be complex and non-linear, but it's often piecewise constant.

Non-linear SVM: The decision boundary is defined by a hyperplane in a higher-dimensional feature space, which can be mapped from the original input space using a kernel function. This allows for more complex and flexible decision boundaries.

Training time:

k-NN: No explicit training phase. Predictions are made on-the-fly based on the distance calculations between the query point and the training points.

Non-linear SVM: Requires training to optimize the parameters (α_i) of the dual problem. This can be computationally expensive for large datasets.

Sensitivity to noise:

k-NN: Can be sensitive to noisy data, especially if the number of neighbors (k) is small. Noisy neighbors can influence the prediction.

Non-linear SVM: Can be more robust to noise, as the kernel function can implicitly smooth out the decision boundary.

Interpretability:

k-NN: More interpretable. The prediction is based on the labels of the nearest neighbors, which can be easily visualized.

Non-linear SVM: Less interpretable, especially when using complex kernel functions. The decision boundary can be difficult to visualize and understand.

Therefore, while both k-NN and non-linear SVM are powerful machine learning algorithms, they differ in their decision boundary formation, training time, sensitivity to noise, and interpretability. The choice of algorithm depends on the specific problem, the size of the dataset, and the desired level of accuracy and interpretability.

2) [3 pts] What is the meaning of n in weighted knn formula and non-linear SVM formula, respectively.

Meaning of the Symbols:

k-NN:

- **w_i:** This symbol represents the weight assigned to the i-th nearest neighbor. In weighted k-NN, not all neighbors are considered equally important. The weight w_i quantifies the influence of the i-th neighbor on the final prediction. This weight is typically inversely proportional to the distance between the query point and the neighbor. Closer neighbors are assigned higher weights, while farther neighbors have lower weights.

Non-linear SVM:

- **α_i:** This symbol represents the Lagrange multiplier associated with the i-th training example. In the context of non-linear SVMs, Lagrange multipliers are used to optimize the objective function during training. The α_i values determine the importance of each training example in defining the decision boundary. Only a subset of training examples, known as support vectors, will have non-zero α_i values. These support vectors are crucial for the SVM model, as they directly influence the position of the decision boundary

7. In 1-nearest neighbor knn with Euclidean distance, the prediction of test data ($h(x)$) is the target value of its single nearest neighbor ($y(x_t)$). In other words,

$$h(x) = y(x_t) \text{ where } x_t = \arg\min_{x \in D} \|x - x_t\|_2 = \arg\min_{x \in D} \|x - x_t\|_2^2$$

1)[6 pts] Change 1-nearest neighbor knn into kernelized 1-nearest neighbor knn. (You can use any kernel function)

Kernelized 1-Nearest Neighbor (K1NN)

In traditional 1-NN, we directly compute the Euclidean distance between the test point and training points in the original feature space. However, in kernelized 1-NN, we implicitly map

the data into a higher-dimensional feature space using a kernel function. This allows us to capture complex non-linear relationships between data points

Differences:

- Traditional 1-NN: Directly compares distances in the original feature space.
- Kernelized 1-NN: Compares similarities in a potentially high-dimensional feature space, implicitly defined by the kernel function.

By using a kernel function, kernelized 1-NN can capture complex patterns and relationships that may not be apparent in the original feature space. However, it's important to note that the choice of kernel function and its hyperparameters can significantly impact the performance of the algorithm.

2) [5 pts] Explain why kernelizing nearest neighbor algorithm is not a very good idea.

While kernelizing the nearest neighbor algorithm can potentially improve performance by capturing complex patterns, it's not always a practical approach. Some of the reasons are:

1. **Computational Cost:**

   Kernel functions, especially those that implicitly map data to high-dimensional spaces, can be computationally expensive to evaluate, especially for large datasets.

   Calculating the kernel similarity between a test point and all training points can be time-consuming, making real-time predictions challenging.

2. **Memory Usage:**

   Storing the entire training dataset can be memory-intensive, particularly for large datasets.

   Kernelized methods often require storing additional information, such as the kernel matrix, which can further increase memory usage.

3. **Overfitting:**

   Kernelized methods can be prone to overfitting, especially when using complex kernels or when the kernel hyperparameters are not tuned carefully.

   Overfitting can lead to poor generalization performance on unseen data.

4. **Lack of Interpretability:**

Kernel methods often operate in high-dimensional feature spaces, making it difficult to interpret the learned models.

This lack of interpretability can hinder understanding and debugging the model.

5. **Sensitivity to Hyperparameters:**

The performance of kernelized methods is highly sensitive to the choice of kernel function and its hyperparameters.

Finding the optimal kernel and hyperparameters can be challenging and time-consuming.

While kernelizing nearest neighbor can be a powerful technique, it's important to weigh the potential benefits against the computational and practical challenges. In many cases, simpler methods like linear nearest neighbor or other more efficient machine learning algorithms may be more suitable.

8. In linear regression, we compute the value of $w = (X^T X)^{-1} X^T y$ (in analytic method). In kernel linear regression, we don't compute w any more.

1) [4 pts] What is the advantage of computing α (not w) ? Explain it USING formulas.

The Advantage of Kernel Trick:

In linear regression, we typically solve for the weight vector w to make predictions:

y = w^T * x

To find the optimal w, we often use methods like Ordinary Least Squares (OLS), which involves solving a system of linear equations. This can be computationally expensive, especially for high-dimensional datasets.

Kernel Trick in Linear Regression

Kernel linear regression leverages the kernel trick to avoid explicitly computing the weight vector w in the high-dimensional feature space. Instead, we compute predictions directly using the kernel function:

$y = \Sigma \alpha_i * K(x_i, x)$

where:

- α_i are the dual coefficients.

- $K(x_i, x)$ is the kernel function, which computes the similarity between the training point x_i and the test point x in a potentially high-dimensional feature space.

Advantages of Kernel Trick:

1. **Implicit Feature Mapping:**

   The kernel function allows us to implicitly map data into a higher-dimensional feature space without explicitly computing the mapping. This can help capture complex patterns that may not be linearly separable in the original feature space.

2. **Computational Efficiency:**

   The kernel trick avoids the explicit computation of the weight vector w, which can be computationally expensive, especially for high-dimensional datasets.

   By directly computing the kernel function, we can often reduce the computational complexity of the prediction process.

3. **Handling Non-Linearity:**

   By using appropriate kernel functions, we can model non-linear relationships between features, which is not possible with traditional linear regression.

In essence, the kernel trick provides a powerful way to extend linear regression to non-linear settings without sacrificing computational efficiency. By working directly with kernel functions, we can often achieve better performance and flexibility.

9. [3 pts] We are going to classify very long text documents and consider each word as an independent feature. Explain why kernel method may not be necessary in this case.

Kernel methods are generally not necessary when classifying very long text documents with words as independent features due to the following reasons:

1. **High-Dimensional Sparse Feature Space:** Text documents represented using a bag-of-words or TF-IDF model already result in a very high-dimensional feature space. Each word corresponds to a unique dimension. In such cases, linear classifiers like SVMs or logistic regression often work well because the decision boundary in this high-dimensional space can effectively separate the data.
2. **Linearly Separable Data:** Text data in high-dimensional spaces is often linearly separable, especially with sufficient data and good preprocessing (e.g., stopword

removal, stemming, TF-IDF weighting). Kernel methods, which are used to map data into even higher-dimensional spaces, are unnecessary if the data is already linearly separable.

3. **Efficiency:** Computing a kernel matrix for very high-dimensional or large datasets (like those generated by long text documents) is computationally expensive and memory-intensive. A linear model avoids this overhead while still achieving good performance.
4. **Interpretability:** Using linear models in the original feature space allows for easier interpretability, such as identifying the most important words (features) contributing to the classification. Kernel methods obscure this interpretability due to their non-linear transformations.

Therefore, the inherent high-dimensional nature of text data and the effectiveness of linear methods make kernel methods unnecessary for this application.

10. [3 pts] Explain the picture in p. 31 in kernel slides.

The image appears to be a slide from a presentation on kernel methods, specifically the Radial Basis Function (RBF) kernel.

The RBF is defined as:

K(xi,xj) = exp(-(||xi-xj||^2) / (2σ^2)) = exp(-γ||xi - xj||^2) for some γ

This is the mathematical formula for the RBF kernel, which is a commonly used kernel function in kernel methods like Support Vector Machines.

The matrix K looks something like: The image shows a 2D heatmap-style visualization of the kernel matrix K, which is a banded diagonal matrix.

K is a banded diagonal matrix: The kernel matrix K has a banded diagonal structure, meaning the non-zero elements are concentrated along the main diagonal and a few diagonals on either side. This structure arises from the properties of the RBF kernel, where the similarity between two data points decreases as their distance increases.

Therefore, this slide is providing an example of the RBF kernel and its corresponding kernel matrix structure, which are important concepts in kernel methods and machine learning.

11. [3 pts] Explain the meaning of formula $l \approx \left(\frac{k}{n}\right)^{1/d}$ in p. 24 in knn slide.

The formula referring to on page 24 of the kNN slide is likely related to the curse of dimensionality concept. It's a common mathematical representation of how distances between points in high-dimensional spaces tend to become less informative.

Curse of Dimensionality

This phenomenon occurs when the number of dimensions (features) in a dataset increases. In high-dimensional spaces, data points tend to become more and more spread out, making it difficult to identify meaningful clusters or patterns.

Mathematical Representation

One way to visualize this is to consider the unit hypercube in d dimensions. The volume of this hypercube is 1^d. As d increases, the volume grows exponentially. However, the surface area of the hypercube grows much faster. This means that most of the volume is concentrated near the surface.

In terms of distances, this implies that most points in high-dimensional space are located near the surface of the space, and the distances between points tend to become more similar. This makes it challenging to distinguish between points based on their distances alone.

kNN relies on the assumption that nearby points are likely to belong to the same class. In low-dimensional spaces, this assumption holds well. However, in high-dimensional spaces, the notion of "nearness" becomes less meaningful. All points become equidistant, making it difficult for kNN to make accurate predictions.

Mitigation Strategies

To address the curse of dimensionality in kNN, several techniques can be employed:

1. Feature Selection: Reduce the number of dimensions by selecting the most relevant features.
2. Dimensionality Reduction: Project the data onto a lower-dimensional space using techniques like PCA or t-SNE.
3. Kernel Methods: Use kernel functions to implicitly map data into a higher-dimensional space where linear separation may be possible.
4. Distance Metrics: Consider using distance metrics that are more suitable for high-dimensional spaces, such as cosine similarity.

By understanding and addressing the curse of dimensionality, we can improve the performance of kNN and other machine learning algorithms in high-dimensional settings

12. [3 pts] Explain the following formula in AdaBoost.

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \times \begin{cases} e^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ e^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

The formula depicted is a key part of the **AdaBoost** algorithm (Adaptive Boosting) and represents the process of updating the weights assigned to training samples in each iteration. Here's an explanation:

Formula:

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \times \begin{cases} e^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ e^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

Components:

1. **$W_i^{(j)}$:**

   The weight of the i-th training sample at iteration j.

   This indicates the importance of the i-th data point when training the current weak classifier.

2. **$Cj(x_i)$:**

   The prediction of the j-th weak classifier for the i-th sample.

3. **y_i:**

   The true label of the i-th training sample.

4. **α_j:**

   A measure of the accuracy of the j-th weak classifier.

   Larger α_j corresponds to more accurate classifiers and increases the effect of their predictions.

5. **Z_j:**

   A normalization factor to ensure the weights $w_i^{j+1)}$ sum to 1 after the update.

6. **Update Rule:**

If the weak classifier predicts the correct label ($C_j(x_i)=y_i$,) the weight for that sample is multiplied by $e^{-\alpha_j}$, reducing its importance.

If the prediction is incorrect ($C_j(x_i)\neq y_i$), the weight is multiplied by e^{α_j}, increasing its importance.

Purpose of the Formula:

- **Focus on Harder Examples:** The weight update process ensures that misclassified samples get higher weights, making them more likely to influence the next weak classifier.
- **Normalization (Z_j):** The term Z_j keeps the weights normalized to sum to 1, ensuring the distribution remains valid for the next iteration.

This iterative reweighting mechanism is a fundamental aspect of AdaBoost, enabling it to combine weak classifiers into a strong ensemble.

13. [3 pts] Explain why boosting is recommended in underfiting problem USING the formula in bias/variance.

Boosting is particularly effective in addressing underfitting problems because it reduces bias while maintaining a controlled variance. This in the context of the **bias-variance tradeoff** using boosting's iterative mechanism.

Bias-Variance Formula:

The expected error of a model can be decomposed as:

$\text{Error}=\text{Bias}^2+\text{Variance}+\text{Irreducible Error}$

Bias refers to the error due to overly simplistic assumptions in the model (underfitting).

Variance refers to sensitivity to fluctuations in the training data (overfitting).

Boosting Reduces Bias in Underfitting Problems:

Boosting focuses on reducing bias by iteratively improving the model in a way that corrects its mistakes. The boosting mechanism in AdaBoost achieves this:

In the AdaBoost formula:

$$\omega_i^{(j+1)} = \frac{wi^{(j)}}{z_j} \times \{ \text{e- } \alpha_{j,} \text{ if } C_j(x_i) = y_i$$

$$\{e^{\alpha j,\ \text{if } Cj}\ (x_i) \neq y_i$$

1.Sequential Learning and Weight Update:

The parameter α_j determines the influence of each weak learner.

By increasing the weights of misclassified samples ($e^{\alpha j}$), boosting ensures that subsequent models pay more attention to previously misclassified points.

This iterative focus on harder samples reduces the overall bias of the combined model.

2.Combining Weak Learners:

Weak classifiers (with high bias individually) are combined into a weighted ensemble, which systematically reduces bias by leveraging their complementary strengths.

The final prediction: $H(x)= \text{sign}\sum_{j=1}^{T} \alpha jCj(x)$

3.**Bias Reduction**:

Boosting converts weak learners with high bias (e.g., simple decision stumps) into a strong learner.

Through iterative reweighting and model combination, the ensemble corrects the underfitting caused by high bias.

Effect on Variance:

- While boosting reduces bias, it does not drastically increase variance because each weak learner focuses only on a subset of the error.
- By keeping individual weak classifiers simple, boosting ensures the overall variance remains manageable, striking a balance between bias and variance.

Reasons why boosting is recommended for underfitting:

- **Underfitting is a high-bias problem**, where the model is too simple to capture patterns in the data.
- Boosting's primary strength is in systematically reducing bias by focusing on the hardest-to-predict samples and refining the model iteratively.

- The controlled variance prevents overfitting, making boosting particularly effective for underfitting scenarios.

Thus, boosting shifts the bias-variance tradeoff toward lower bias while maintaining a reasonable variance, improving performance in underfitting problems.

14. Explain the formulas on p. 22 in boosting slide with your own words.

1) [4 pts] Explain why error function $L = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + h_t(x_i))$ is approximated to $\underset{h_t}{\operatorname{argmin}} \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i h_t(x_i) \right]$

The error function L given in the slides represents the loss between the predicted and actual values across all instances in the dataset. The term $\hat{y}_i^{(t^{-1})}$) is the prediction from the previous iteration of the boosting process, and $h_t(x_i)$ is the prediction of the new weak learner being added at iteration t.

This formula suggests that in each iteration of gradient boosting, we're looking to add a weak learner h_t that, when combined with the predictions from the previous iteration $\hat{y}_i^{(t^{-1})}$, minimizes the overall loss. The function L thus incorporates both the current state of the ensemble's predictions and the potential improvement from adding h_t. This form allows the boosting algorithm to focus incrementally on the areas where previous iterations were lacking, effectively reducing the residual errors.

2) [4 pts] Explain why $\underset{h_t}{\operatorname{argmin}} \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i h_t(x_i) \right]$ is approximated to the following.

$$\underset{h_t}{\operatorname{argmin}} \sum_{i=1}^{n} [g_i h_t(x_i)] + \text{const}$$

$$g_i = \frac{\partial l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}$$

The function $g_i = \frac{\partial L\left(y_i, \hat{y}_i^{(t-1)}\right)}{\partial \hat{y}_i^{(t-1)}}$ represents the gradient of the loss function at each data point, which indicates the direction and magnitude of the error. This gradient tells us how much changing the prediction $\hat{y}_i^{(t-1)}$ would change the loss, which is crucial for improving the model.

By using the gradient in the approximation, the boosting algorithm seeks to find a weak learner h_t that best reduces the loss as indicated by the gradient. The term "const" reflects components of the loss that are independent of h_t, and therefore do not affect the optimization process.

3) [4 pts] Explain why new weak classifier $h_t(x_i)$ should be the one that minimizes the residual error from previous ensemble. Explain it USING formulas (you can use the example of regression with squared error).

In the context of boosting, particularly with the squared error loss, the residual error $r_i = y_i - \hat{y}^{(t-1)}$ is the difference between the actual and the predicted values from the previous iteration. The goal of adding a new weak learner hth_tht is to correct these residuals.

If we consider the squared error $L(y_i, \hat{y}_i^{(t-1)} = \left(y_i - \hat{y}^{(t-1)}\right)^2$, its gradient with respect to the prediction $\hat{y}_i^{(t-1)}$ is $-2\left(y_i - \hat{y}^{(t-1)}\right)$, which is simply $-2r_i$. Minimizing this gradient (or equivalently, minimizing r_i) directly minimizes the squared error. Thus, h_t should be selected to predict the negative gradient (i.e., the residuals), as doing so will most effectively reduce the overall prediction error in the subsequent iterations.

Therefore, by focusing on reducing the residuals or correcting the most significant sources of error from the previous iteration, boosting incrementally improves the model's accuracy, thereby effectively addressing areas where the model is still underfitting the data.

15. [3 pts/ea] Suppose, in your algorithm, training error is low, but test error is high. You are going to fix this problem. Explain the following cases.

1) Increasing the size of training data will help?

When training error is low but test error is high, the model is likely overfitting the training data. Overfitting occurs when the model learns patterns specific to the training set that do not generalize well to unseen data.

Yes, increasing the size of the training data can help reduce overfitting in many cases.

1. **Improved Generalization**:

   With more training data, the model is exposed to a broader range of examples and patterns. This reduces the likelihood of memorizing noise or overly specific details in the training set, helping the model generalize better to new data.

2. **Bias and Variance Tradeoff**:

Overfitting corresponds to high variance in the bias-variance tradeoff. Adding more training data reduces variance by stabilizing the learning process and providing the model with more representative information about the underlying data distribution.

3. **Mitigating the Effects of Noise**:

If the training set is small, the model might overfit to noisy or outlier examples. With a larger dataset, the impact of noisy samples is diluted, and the model learns more robust patterns.

Situations When This May Not Help:

- If the model is too complex (e.g., deep neural networks with too many parameters for the dataset size), simply increasing the training data might not fully resolve the problem. Additional strategies like regularization or simplifying the model may also be needed.
- If the training data does not adequately represent the test data distribution, increasing its size will not solve the problem (this is a data quality issue, not a quantity issue).

Therefore, increasing the size of the training data is a reasonable and often effective approach to improve generalization and reduce test error in cases of overfitting. However, its success depends on having diverse and representative data

2) Feature selection methods will help?

Yes, feature selection methods can help reduce test error in some cases, especially if the high test error is due to irrelevant or redundant features.

1. **Reducing Overfitting:**

Irrelevant or redundant features can introduce noise into the model, making it more prone to overfitting. The model may "learn" patterns from these irrelevant features that do not generalize to the test data.

By selecting only the most important and relevant features, feature selection methods simplify the model, reducing its capacity to memorize noise and improving generalization to unseen data.

2. **Dimensionality Reduction:**

If the dataset has a high number of features relative to the number of training samples (high-dimensional data), the model might suffer from the curse of dimensionality. Feature

selection reduces the dimensionality, making it easier for the model to find meaningful patterns without overfitting.

3. **Improved Interpretability:**

   By removing irrelevant features, the resulting model becomes simpler and more interpretable. This can help identify which features contribute to the problem and which ones might be unnecessary.

Types of Feature Selection:

- **Filter Methods:** Use statistical measures (e.g., correlation, mutual information) to rank and select features.
- **Wrapper Methods:** Use the performance of the model (e.g., via cross-validation) to evaluate subsets of features.
- **Embedded Methods:** Incorporate feature selection into the model training process (e.g., LASSO regression, tree-based feature importance).

Situations When Feature Selection May Not Help:

- If all features are relevant and contribute to the predictive power of the model, removing features might degrade performance.
- If the model already has inherent feature selection mechanisms (e.g., tree-based models like Random Forest), additional feature selection may not provide significant benefits.

Feature selection methods are helpful when high test error is caused by irrelevant, redundant, or noisy features. By focusing on the most relevant features, these methods reduce overfitting, simplify the model, and improve generalization. However, if all features are relevant, feature selection might not help and could even harm performance.

3) Using cross-validation will help? Explain using the formula of bias/variance.

Yes, using cross-validation can help identify and mitigate the issue of high test error (overfitting) by providing a better estimate of model performance and helping to tune the model's complexity. Here's why, explained with the bias-variance tradeoff formula.

Formula for Error Decomposition:

Error=Bias^2+Variance+Irreducible Error

- **Bias** measures how much the model's predictions deviate from the true values due to overly simplistic assumptions (underfitting).
- **Variance** measures how much the model's predictions vary across different training sets due to over-sensitivity to the training data (overfitting).

Cross-Validation Helps as follows:

Cross-validation is a technique for evaluating the model's generalization error by splitting the data into multiple training and validation subsets. Its role in addressing test error stems from its ability to find the right balance between bias and variance.

1.Reduces Variance in Model Selection:

By splitting the data into multiple folds and averaging the performance across these folds, cross-validation provides a more robust estimate of the model's true performance compared to a single train-test split.

When tuning hyperparameters (e.g., model complexity, regularization), cross-validation ensures that the selected model generalizes better, thus reducing overfitting (variance).

2. Detecting Overfitting:

If training error is low but validation error (in cross-validation) is high, this signals overfitting.

Cross-validation identifies this issue early by evaluating the model on unseen validation data during training, preventing reliance solely on the training set.

3. Bias-Variance Perspective:

Cross-validation does not directly reduce bias, but it helps find the optimal tradeoff between bias and variance:

Models with high variance (complex models) will show fluctuating errors across folds, indicating overfitting.

Simpler models with higher bias (underfitting) will have consistently higher validation errors.

By selecting the model with the lowest average validation error, cross-validation helps balance bias and variance to minimize test error.

4. Prevents Data Leakage:

Ensures that no test data is inadvertently used during training, making the error estimates more reliable and representative of true generalization error.

When Cross-Validation Might Not Help:

- Cross-validation only evaluates the model; it does not directly fix the underlying issue (e.g., noisy features, insufficient data, etc.).
- If the training data is not representative of the test data (distribution shift), cross-validation will still suffer from the same limitations.

Therefore, Cross-validation is a critical tool for diagnosing and reducing overfitting by providing a better estimate of the generalization error. Using cross-validation allows you to select a model with an appropriate balance between bias and variance, which in turn reduces test error and improves overall performance

4) Boosting will help? If boosting is helpful, how exactly are you going to use boosting in this scenario?

Yes, boosting can help when training error is low but test error is high, which indicates overfitting to the training data or poor generalization. Boosting reduces test error by iteratively improving model performance, focusing on difficult-to-predict samples, and combining weak learners into a strong ensemble. However, how boosting is applied depends on the specific context.

Boosting can help in the follows ways:

1. **Reducing Bias:**

   If the test error is high because the weak learner is underfitting (too simple), boosting improves performance by sequentially building stronger models with low bias. Each weak learner in boosting focuses on the errors of the previous ones.

2. **Controlling Overfitting:**

   Although boosting iteratively focuses on harder-to-predict samples, the way it combines weak learners typically does not increase variance excessively. By keeping individual learners simple, boosting controls overfitting.

3. **Adaptability:**

Boosting adjusts dynamically to emphasize difficult samples without requiring changes to the underlying weak learner.

Using Boosting in This Scenario:

1. **Choose the Right Boosting Algorithm:**

Use a boosting algorithm like AdaBoost (if the problem involves decision stumps or classifiers) or Gradient Boosting Machines (GBMs) (if using regression trees or more complex learners).

Modern implementations like XGBoost, LightGBM, or CatBoost are highly optimized for practical use.

2. **Steps to Apply Boosting:**

Step 1: Train an Initial Weak Learner: Start with a simple weak learner, such as decision stumps or shallow trees, on the training data.

Step 2: Update Sample Weights: Use the AdaBoost weight update formula:

Step 3: Repeat: Iteratively train new weak learners on the reweighted data to correct errors from previous learners.

Step 4: Combine Weak Learners: Use a weighted combination of weak learners to form the final model

3. **Tuning Boosting Parameters:**

Set parameters like the number of weak learners (T) and learning rate (η) carefully:

A smaller learning rate (η) often improves generalization by slowing down the boosting process.

Use cross-validation to tune these parameters and avoid overfitting.

4. **Regularization:**

Add regularization techniques (e.g., limiting tree depth or feature sampling) if the boosting algorithm overfits on the training data.

5. **Monitor Overfitting:**

Regularly evaluate training and validation errors to ensure the boosting algorithm isn't overly fitting the noise in the data.

When Boosting May Not Work:

Insufficient or poor-quality data: If the dataset is small or noisy, boosting may amplify the noise rather than the signal.

Severe Overfitting: If the test error is very high due to extreme overfitting, simpler models (or better data preprocessing) may be needed before boosting.

Therefore, boosting is highly effective in reducing both training and test errors, especially for overfitting or underfitting scenarios. In this case, boosting can reduce test error by focusing on difficult-to-predict examples and combining multiple weak learners. To ensure success, we have to tune the algorithm using cross-validation and monitor for overfitting during training.