CS7641 ML Lecture Summaries
Chapter 1: Supervised Learning

Kyle Nakamura

1.1 Decision Trees

1.1.1 Difference between Classification and Regression

Today's lecture focuses on supervised learning, specifically classification and regression. Classification involves mapping inputs to discrete labels, such as determining if a picture is of a male or female. Regression, on the other hand, involves mapping inputs to continuous values, like mapping pictures to the length of someone's hair. The lecture provides examples and emphasizes the importance of understanding the difference between classification and regression.

1.1.2 Classification or Regression Question

In this lecture on decision trees, the instructor emphasizes the importance of understanding the difference between classification and regression in supervised learning. The students are given a quiz with three questions to test their understanding. The first question deals with credit history as the input, including factors such as the number of loans, income, defaults, and late payments. The output of the learning algorithm in this scenario is whether a person should be lent money or not. The students are asked to determine if this is a classification or regression task. The second question involves taking a picture as input and the output is to classify the age of the person as high school age, college age, or grad student age. The third question is similar, where the input is also a picture, but this time the output is the actual age of the person. Again, the students must determine if these are classification or regression tasks.

1.1.3 Classification or Regression Solution

In this lecture, the professor explains the difference between classification and regression tasks in machine learning. They clarify that whether a task is classified as classification or regression depends on the nature of the output, not the input. If the output is a continuous quantity, it is considered a regression task, while if the output is from a discrete set, it is a classification task. The professor gives examples to illustrate this. In the first example, predicting whether to lend money or not, the output is binary (yes or no), making it a classification problem. In the second example, categorizing individuals into high school, college, or graduate student, the output is from a small discrete set, making it another classification problem. In the third example, predicting ages, although the output has a wider range, including fractional values, it is considered a regression problem because it is a continuous quantity. However, the professor points out that the choice between classification and regression can vary depending on how the output is defined and argues that in practice, it is often easier to define it as a real number and consider it a regression task. The lecture concludes by suggesting a further exploration of the definitions of classification and regression problems.

1.1.4 Classification Learning One

In this lecture, the instructor defines two key terms in classification learning: instances and concepts. Instances refer to the input data, such as pictures or credit scores, that describe the input space. Concepts, on the other hand, are the functions that map these instances to specific outputs, such as true or false. The instructor explains that concepts are a formal way to describe sets of things and are essentially a mapping between objects in the world and membership in a set. The lecture emphasizes that concepts can be applied to various classification scenarios, including binary or multi-class classification.

1.1.5 Classification Learning Two

In this lecture on Classification Learning Two, the instructor discusses the concept of a target concept, which is the specific function that we are trying to find. The target concept is important because, although we may have a general notion in our heads, we need to have it written down somewhere to know if it is right or wrong. The instructor explains that in order to teach someone a concept, we need to convey it to them. This involves understanding the notions of instances (input), concepts (which map input to output), and hypothesis class (the set of functions we are willing to consider). The hypothesis class is important because it allows us to restrict the search for the right function, which is necessary when working with finite data. The instructor mentions that initially, the hypothesis class can be thought of as all possible functions, but in the context of classification learning, we are already restricting ourselves to a subset of functions. The instructor emphasizes the use of samples or training sets to determine the right answer in machine learning.

1.1.6 Classification Learning Three

A training set is a set of inputs paired with their corresponding labels, which are the correct outputs. The training set is used to determine the correct concept or function. Inductive learning involves providing lots of examples and labels to explain the target concept, rather than giving a precise definition. A candidate is a concept that is hypothesized to be the target concept. The testing set is used to evaluate the performance of the candidate concept by comparing its outputs to the actual labels. The training set and testing set should not be the same in order to test the ability to generalize. Generalization, which is the ability to apply concepts in new examples, is the goal of machine learning.

1.1.7 Example 1 Dating

The lecture is about decision trees in the context of supervised learning. The speaker introduces the concept using an example of deciding whether to enter a restaurant on a date. The problem is a classification problem with binary outcomes: enter or move on to the next restaurant. The speaker discusses the need to define the attributes or features that define a restaurant, such as the type of cuisine, atmosphere, occupancy, and the importance of impressing a date. The first two attributes have multiple possible categories, while the last two are binary. The lecture ends with the suggestion to explore additional features and discuss how to solve the problem.

1.1.8 Representation

In this lecture on decision trees, the speaker discusses the representation of decision trees and their components. The speaker mentions that decision trees are made up of nodes and edges, where nodes represent attributes and edges represent the possible values of those attributes. The speaker gives an example of a decision tree where the attribute is whether the person is hungry or not, and the possible values are yes or no. The speaker explains that decision trees allow for a series of questions to be asked, leading to a final output answer. The speaker also mentions that decision trees can include attributes that are unrelated to the main topic, but may still be important. Overall, the lecture introduces decision trees and their representation, with examples provided to aid understanding.

1.1.9 Representation Quiz Question

In this lecture, the instructor presents a concrete example of a decision tree. The decision tree considers features such as restaurant occupancy, type of restaurant, and whether the people inside look happy. The possible outputs are either to go or not go into the restaurant. The instructor provides a table with six features and specific values for each. The task is to determine the output of the decision tree for each case. The decision tree is presented as a classifier or concept that can be used to make predictions. The lecture concludes with the instructor challenging the students to trace through the decision tree and determine the class for each situation.

1.1.10 Representation Quiz Solution

Decision trees ask a series of questions to determine the output. The path of the tree is determined by the values of the features. It is important to start at the root of the tree and ask the questions in a specific order. In the given example, only the attributes ”occupied,” ”type,” and ”happiness” are considered, while ”hot date,” ”hungry,” and ”raining” are irrelevant. The tree helps determine whether to go or not go to a restaurant based on these features. The tree is considered a candidate concept for testing set examples. There are different possible trees, but in this case, the specific tree with these features and questions is chosen. The next topic will be deciding whether to choose this tree over other possible trees.

1.1 Decision Trees - 1.1.11 Example 2 - 20 Questions To build decision trees from training data, the lecturer proposes playing a game of 20 questions. The goal of the game is to guess a famous person by asking yes/no questions. The lecturer and Michael engage in the game, with Michael asking questions and the lecturer answering. Through the game, they discover that the questions should be asked in a specific order to effectively narrow down the possibilities and make progress towards finding the correct answer. The lecturer emphasizes that the usefulness of a question depends on the answers received in previous questions. They conclude that this idea forms the basis of an algorithm for building decision trees.

1.1.12 Decision Trees Learning

In this lecture on decision trees, the lecturer discusses the process of using questions to narrow down and find the desired answer. Inspired by the game "20 Questions," the lecturer explains that the first step is to imagine possible answers and then to think of a question that would divide them into roughly equal groups. This process can be translated into supervised learning and classification, where a training set is used to determine the best attribute to split the data. The lecturer emphasizes that the definition of "best" is to split the data roughly in half. Once an attribute is chosen, a question is asked, and depending on the answer, another attribute is selected. This process is repeated until the possibilities are narrowed down to a single item, which is the desired answer. The lecturer describes this as an algorithm for building a decision tree. The only difference between playing the game interactively and learning a decision tree is that in the latter, all possible paths and attributes must be considered since the answer is not known in advance. The lecturer concludes by suggesting using pictures to visualize the decision tree and evaluating the definition of "best."

1.1.13 Best Attribute Quiz Question

In this lecture on decision trees, the concept of a "best attribute" is discussed and defined more precisely. Three attributes are presented, with instances labeled either with red x or green o. In the first case, instances are sorted into two piles, with some xs and some os on both sides. In the second case, all instances are sorted to the left. In the third case, xs are sorted to the left and os to the right. The task is to rank these attributes from best to least best.

1.1.14 Best Attribute Quiz Solution

In this lecture on decision trees, the speaker and Michael discuss the best attribute to split on and assign rankings. They agree that the third attribute is the best because it separates the data into two distinct groups. They also conclude that the second attribute is the worst as it does not contribute any meaningful splitting. The discussion then shifts to the first attribute, which they consider to be in between in terms of usefulness. They analyze the distribution of red and green data points before and after splitting on this attribute. They debate whether this attribute provides any valuable information or leads to overfitting. Ultimately, they conclude that both the first and second attributes are not ideal, but it is difficult to determine which one is worse. The lecture ends with a lighthearted remark likening the attributes to the House of Representatives and the Senate, although the meaning remains unclear.

1.1.15 Decision Trees Expressiveness AND

In this lecture on decision trees, the speaker discusses the expressiveness of decision trees by looking at Boolean functions, specifically the And function. The speaker and Michael discuss how to build a decision tree to represent the And function. They go through the process step by step, starting with the attributes A and B and determining the true and false outcomes for each attribute. The speaker highlights that the order of the attributes doesn't matter in this case, as And is a commutative function. The lecture concludes by noting that decision trees can effectively represent the Boolean function And.

1.1.16 Decision Trees Expressiveness OR

In this lecture on decision trees, the instructor discusses the expressiveness of decision trees using the Boolean function A or B as an example. Initially, the instructor attempts to use the same approach as in the previous lecture where A and B were used. However, this approach is not effective for A or B. The instructor decides to start from scratch and splits on attribute B. If B is true, the output is set as true, representing the case when A or B is true. If B is false, the instructor splits on attribute A. If A is true, the output is true, representing the case when A or B is true. If A is false, the output is false, representing that A or B is false. The instructor confirms that this representation correctly represents A or B. Additionally, the instructor swaps the positions of A and B to test if it still works, and confirms that the two decision trees mirror each other.

1.1.17 Decision Trees Expressiveness XOR

The lecture discusses the concept of XOR (exclusive or) and its representation using a decision tree. XOR is true if either A or B is true, but not if both are true. The lecturer provides real-life examples and explains how XOR can be represented using a decision tree. The decision tree for XOR involves splitting on A and B, with separate branches for all possible inputs. The lecturer also compares the decision tree representation of XOR to the truth table representation and explains that for simpler functions like XOR, the decision tree can be simplified by eliminating unnecessary branches. Finally, the lecturer mentions that the difference between representing XOR using the entire truth table and using a decision tree will not matter when dealing with more complex functions or attributes.

1.1.18 Decision Tree Expressiveness

In the lecture on decision tree expressiveness, the instructor discusses the complexity of decision trees for generalized versions of OR and XOR. For n attributes, the decision tree for the n version of OR has n nodes, making it linear in size. On the other hand, the decision tree for the n version of XOR has approximately 2^n nodes, which is exponential in size. The XOR problem is considered hard and difficult to solve because it requires examining every attribute. The instructor suggests that it is better to have problems more like OR (relatively easy) than XOR. Additionally, the instructor mentions that finding a good representation for the problem, such as creating a new variable that simplifies the decision tree, can be considered "cheating" but can also lead to more effective machine learning.

1.1.19 Decision Tree Expressiveness Quiz Question

The lecture discusses the expressiveness of decision trees in representing different functions and the number of possible decision trees that need to be considered. The lecturer uses the example of XOR and OR functions to illustrate the difference in complexity. The key question is how many decision trees need to be examined to find the right one. The lecturer asks the student to determine the number of decision trees for a given number of Boolean attributes. The student answers that there are a large number of decision trees, with n factorial possibilities for nodes and an exponential number of leaves. The lecturer agrees with the student's analysis and explains that a truth table can be used to represent the possible input-output combinations for Boolean functions. The lecturer concludes by asking the student to determine the number of rows in the truth table.

1.1.20 Decision Tree Expressiveness Quiz Solution

The number of rows in a decision tree is determined by the number of attributes being split on. If there is only one variable being split on, there are two rows. For two variables, there are four combinations, and for three variables, there are eight combinations. In general, with n attributes, there are 2 to the power of n different rows. This is always the case when dealing with n boolean attributes. However, this only accounts for the number of rows in the truth table. The size of the truth table itself is another question that needs to be addressed.

1.1.21 Decision Tree Expressiveness Quiz 2 Question

The question is about the number of ways in which a column of outputs can be filled out given 2^N rows. The task is to determine the number of different decision trees and functions that can be derived from the given instances. The outputs can either be true or false.

1.1.22 Decision Tree Expressiveness Quiz 2 Solution

The lecture discusses the expressiveness of decision trees and the exponential growth in the number of possible decision trees as the number of positions increases. The speaker explains that each empty box in a decision tree represents a bit, and the number of different bit patterns in a decision tree is 2 to the power of 2 to the power of n. This results in a very large number, even for small values of n. For example, for n = 6, the number of possible decision trees is an extremely large number. This exponential growth highlights the need for a clever search algorithm to efficiently explore the hypothesis space of decision trees. Otherwise, the search process would involve checking billions of possible decision trees.

1.1.23 ID3

The lecture discusses the ID3 algorithm for building decision trees. The algorithm involves selecting the best attribute to split the data and creating branches based on the possible attribute values. The lecture focuses on defining what is meant by the ”best” attribute, which is commonly determined using information gain. Information gain measures the reduction in randomness or entropy in the labels of the data when a particular attribute value is known. The lecture explains that entropy is a measure of randomness and provides examples to illustrate the concept. The formula for entropy is also mentioned. The goal in the ID3 algorithm is to choose the attribute with the maximum information gain, as it provides the most useful information for classification.

1.1.24 ID3 Bias

The lecture discusses the concept of inductive bias in the context of decision trees and specifically focuses on the ID3 algorithm. Inductive bias refers to the preferences and restrictions imposed on the search space when using algorithms. There are two types of biases: restriction bias and preference bias. The restriction bias in this case is limited to considering only decision trees as the hypothesis set. The lecture emphasizes that this restricts the algorithm from considering other functions or equations. The preference bias, on the other hand, determines the preference for certain decision trees over others. ID3 algorithm's inductive bias prefers decision trees that have good splits near the top, even if multiple trees can represent the same function. Additionally, the algorithm prefers decision trees that better model the data and give correct answers. The lecture also notes that ID3 tends to prefer shorter trees over longer ones, as shorter trees with good splits can lead to faster convergence.

1.1.25 Decision Trees Continuous Attributes

Decision trees have been explored extensively, but there are still some open questions to consider. One question is how to handle continuous attributes in decision trees. One approach is to create a branching factor equal to the number of possible values for the continuous attribute, such as age. Another possibility is to only include values from the training set, but this approach raises issues when encountering new values in the future. Ranges can also be used, with a decision tree representing a range of values, such as ages in the 20s. Evaluating attributes like this can be done using a formula from ID3. However, there are numerous possible attributes to check, especially if it is a truly continuous variable with an infinite number of values. To address this, one can use the training set to select questions that cover the data in the training set. For example, if all values are in the 20s, there is no need to ask the question. Instead, one can split based on values less than or greater than a certain threshold. Various methods, such as binary search, can be used to find appropriate threshold values.

1.1.26 Decision Trees Other Considerations Quiz Question

The question being asked is whether it makes sense to repeat an attribute along a specific path in a decision tree. The context clarifies that the repetition being referred to is within the same path, rather than in different parts of the tree. The lecturer gives an example where an attribute A appears twice in the tree, but not along the same path, meaning that once a related attribute B is answered, the question about A will only be asked once. The students are asked to answer the true/false question.

1.1.27 Decision Trees Other Considerations Quiz Solution

In decision trees, it does not make sense to repeat an attribute along a path in the tree for discrete value trees. This is because information gain will automatically pick the attribute that has not been split on before, as splitting on an already split attribute will not provide any additional information. However, for continuous attributes, it does make sense to ask a different question about the same attribute later in the tree. For example, if we ask if age is in the 20s or not, we are actually creating a discrete-valued attribute for the decision tree. Asking a different question, such as if age is less than 20 or greater than 40, does make sense.

1.1.28 Decision Trees Other Considerations

The lecture discusses considerations for decision trees, specifically in relation to overfitting and noise in the data. Overfitting occurs when the decision tree is too large or complex, violating Occam's Razor. To avoid overfitting, one approach is to use cross-validation and select the tree with the lowest error on the validation set. Another approach is to use a more efficient method of validation, where the tree is expanded until the error on the validation set is low enough. In terms of tree expansion, breadth-first expansion is suggested as a way to prevent favoring one side of the tree over the other. Pruning is another technique to address overfitting, where the tree is built first and then leaves are collapsed based on the error on the validation set. Pruning is a simple addition to the ID3 algorithm.

1.1.29 Decision Trees Other Considerations Regression

One consideration when using decision trees is how to handle regression problems, where the outputs are continuous rather than discrete. The usual measure of information gain may not work well for continuous values, so alternative measures like variance could be used. The splitting criteria and what to do in the leaves of the tree are also important considerations. Different fitting algorithms can be used in the leaves, such as reporting the average or performing a linear fit. Pruning can also be used to improve the tree. When dealing with regression, the idea of voting, such as taking the average, can be applied.

1.1.30 Decision Trees Wrap up

In this lecture on Decision Trees, the speaker reviews the main concepts covered. These include the Decision Tree representation, the top-down algorithm for inducing a Decision Tree (called ID3), the expressiveness and bias of Decision Trees, and the method of using information gain to decide on splits. The problem of overfitting in Decision Trees is also discussed, along with strategies for pruning the tree to avoid overfitting. The speaker concludes by acknowledging that there is still more to learn about Decision Trees, which students will explore through assignments.

1.1.31 What is Regression Question

The lecture discusses regression in the context of supervised learning, where examples of inputs and outputs are used to predict new inputs' corresponding outputs. Regression specifically focuses on mapping continuous inputs to outputs. The term "regression" is used to describe this mapping, although it is unrelated to its psychological definition. The lecture provides an example of predicting the average height of a person's children based on their height.

1.1.32 What is Regression Solution

Regression to the mean is a phenomenon observed when measuring people's heights and the heights of their children. It is found that very tall individuals tend to have children who are taller than average but still closer to the average height. This regression towards the mean can be understood as a drifting back towards the average height of the population, like a random walk. This concept explains why everyone does not end up being the same height.

1.1.33 Regression and Function Approximation

Regression and function approximation are connected through the concept of regression to the mean. The relationship between parent height and average child height can be represented on a graph, with the slope of the line being less than one (specifically, two thirds). This means that children of taller parents are shorter than their parents, while children of shorter parents are taller. This regression to the mean was observed in the late 1800s and led to the development of the term "regression" in statistics. However, the term is now used to refer to approximating a mathematical relationship based on data points, rather than the original meaning of regression to the mean. This misinterpretation of terminology can be seen in other fields, such as reinforcement learning, where the term doesn't align with its original definition.

1.1.34 Linear Regression

The lecture discusses the concept of regression in machine learning and its relevance in understanding various topics. The speaker uses the example of housing prices to explain regression. They present a graph showing the relationship between house size (square footage) and cost. They plot nine houses and show how the price tends to increase with size. They then ask the question of what a fair price would be for a 5,000 square foot house. They emphasize the need to interpolate or estimate based on the available data points. The speaker introduces the idea of finding a linear function that best captures the relationship between size and cost. They show the best linear function that minimizes squared error and explain how it balances all the errors. They note that this particular line predicts a cost of around $4,000 for a 5,000 square foot house, which may not seem ideal but still captures the increasing cost trend with size.

1.1.35 Find the Best Fit Question

In this lecture on decision trees, the speaker discusses the problem of finding the best fit line. They explain that there are an infinite number of lines to choose from, and the goal is to find the one that minimizes the squared error between the line and a set of points. The speaker raises the question of how to solve this problem, suggesting options such as hill climbing, calculus, random search, or seeking help from a physicist due to the involvement of continuous quantities.

1.1.36 Find the Best Fit Solution

In this lecture, the instructor explains how to find the best fit solution using calculus. They demonstrate this using an example of finding the best constant function for a given set of data points. The error function used in this example is the sum of squared differences between the chosen constant and the actual values. The instructor explains that there are different error functions that can be used, but the squared error function is particularly well-behaved and allows for the use of calculus to find the minimum error value. The derivative of the error function is computed using the chain rule, and setting it equal to zero allows for finding the minimum value. Solving the equation gives the best constant as the average of all the y-values, also known as the mean. The instructor mentions that the squared error function is convenient because it brings the mean into the picture and it generalizes to higher order functions and more variables like lines with non-constant slope.

1.1.37 Order of Polynomial

In this lecture, the professor discusses the concept of fitting functions to data points. They mention that in addition to finding the best line, it is also possible to find the best constant and the best parabola to fit a given set of points. The professor emphasizes that the choice of the degree of the polynomial function used for fitting depends on the nature of the data. They demonstrate the fitting process for a parabola and show that as the degree of the polynomial increases, the error in fitting the points decreases. However, they also point out that high-degree polynomials may result in curves that appear unrealistic or erratic for certain data points. The professor concludes by showing a plot of the squared error for the best fit at each degree of the polynomial, demonstrating that as the degree increases, the error approaches zero.

1.1.38 Pick the Degree Question

The lecturer presents a quiz on selecting the degree for the housing data. The choices are degree zero (constant), degree one (line with increasing slope), degree two (parabola with leveling off), degree three (cubic with flattening and rising), and degree eight (octic with perfect fit to data points). The lecturer also mentions the term "full monty" to refer to the octic degree.

1.1.39 Pick the Degree Solution

The lecturer discusses how to choose the degree for a solution in decision tree models. They suggest that a degree of 3 is the optimal choice based on the given data. They explain that lower degrees make more errors, while higher degrees overfit and make unnecessary adjustments to match training data. The lecturer highlights that the cubic degree solution clings closely to the points but stays between them, which is seen as a smart approach. The lecturer concludes by mentioning the need to evaluate this choice more concretely.

1.1.40 Polynomial Regression

In this lecture on Polynomial Regression, the focus is on fitting data to a polynomial function. The goal is to find coefficients that minimize the difference between the polynomial and the data points. The lecture explains how this can be done using matrix operations. The process involves arranging the data into a matrix and defining the polynomial coefficients as variables. The matrix is multiplied by its transpose and the result is inverted. Then, the inverted matrix is multiplied by the transpose of the original matrix and then by the vector of y-values. This computation yields the desired coefficients for the polynomial regression. It is mentioned that the matrix operation has nice properties in terms of being invertible and minimizing the least squares. However, the lecture does not delve into why this is the case, leaving that discussion for future reference to calculus. The important takeaway is that the process involves representing the data in matrix form, performing the outlined computations, and obtaining the polynomial coefficients required for polynomial regression.

1.1.41 Errors Question

The lecture discusses the need for solving problems using regression techniques instead of linear equations due to the presence of errors in the training data. The errors can arise from sensor errors, malicious intent, transcription errors, or un-modeled influences. Examples of un-modeled influences are changes in housing, location, quality, builder, colors, time of day, and interest rates. The lecturer asks students to identify the important factors that can affect machine learning and regression.

1.1.42 Errors Solution

In order to accurately fit our data and avoid fitting the errors, it is important to understand the underlying signal. We need to determine the underlying function without the influence of errors and noise.

1.1.43 Cross Validation

Cross validation is a technique used in machine learning to evaluate the performance of different models. The goal is to find a model that is complex enough to capture the structure of the training data, but not overly complex that it fails to generalize well to new, unseen data. The traditional approach of using a separate test set for evaluation can be seen as ”cheating” because it directly fits the errors in the test set, rather than capturing the true structure of the data. To address this issue, cross validation uses a portion of the training data as a substitute for the test set. This ”cross validation set” is used to evaluate different models and select the one with the lowest error. The training data is split into ”folds” and the models are trained on different combinations of folds, with one fold left out as the cross validation set. This process is repeated multiple times, each time using a different fold as the cross validation set. The errors obtained from each iteration are then averaged to obtain the overall performance measure. The model class, such as the degree of the polynomial, that achieves the lowest error on the cross validation set is chosen as the final model. This approach ensures that the selected model both captures the structure of the data and generalizes well to new data.

1.1.44 Housing Example Revisited

In this lecture on decision trees and housing examples, the instructor discusses the relationship between the degree of a polynomial and the training error. As the degree increases, the error decreases, indicating a better fit to the data. However, when using cross-validation, which involves splitting the data into chunks and predicting each chunk using the rest of the data, the cross-validation error initially starts higher than the training error. This is because the models are trained to minimize error on the training set and therefore may have more error on the unseen data. As the degree of the polynomial increases, the ability to fit the data improves, but the cross-validation error also increases. This demonstrates the tradeoff between overfitting and underfitting the data. Too few degrees of freedom result in underfitting, while too many degrees of freedom can lead to overfitting. The ideal model is one that fits the data without overfitting or underfitting. In the housing example, a degree of three fits the data the best, with a degree of four barely using the extra degree of freedom and performing slightly worse in terms of generalization.

1.1.45 Other Input Spaces

In this lecture, the instructor discusses the use of vector inputs in regression. They give an example of predicting housing costs using two input variables: size and distance to the nearest zoo. They explain that combining these two variables in a linear fashion can help predict the cost. The instructor also mentions the importance of considering discrete variables, such as whether someone has a job or the value of their assets. They explain that these variables can be represented as numbers, either as Boolean variables (0 or 1) or by assigning numerical values. They mention the RGB color encoding as an example of assigning numerical values to discrete variables. The lecture concludes by acknowledging that these issues will be further explored when encoding problems as machine learning problems.

1.1.46 Conclusion

The lecture covered various aspects of regression, including historical facts, model selection, overfitting and underfitting, cross-validation, linear and polynomial regression, the best constant for squared error (which is the mean), representation in regression, and issues with input representations.

1.1.47 Neural Networks

Neural networks are modeled after the structure of neurons in the brain. They consist of a network of interconnected computational units called neurons. Each neuron has a main part called the cell body, an axon which transmits electrical impulses, and synapses which connect to other neurons. Neural networks can be tuned or changed to fire under different conditions and compute different things. In artificial neural networks, a simplified version of a neuron is used. This involves multiplying inputs by corresponding weights, summing them up, and comparing the sum to a firing threshold to determine the output. This simplified model is called a Perceptron. Networks of Perceptrons can compute various functions.

1.1.48 Artificial Neural Networks Question

In this lecture, the concept of an artificial neuron and its functioning is explained through an example. The input to the neuron is given as 1, 0, and -1. The corresponding weights for these inputs are 0.5, 0.6, and 1. The lecture also mentions that the threshold is set at 0, meaning that the neuron will fire if the weighted sum is above or equal to 0. The task is to compute the output y based on these numbers.

1.1.49 Artificial Neural Networks Solution

In this lecture on artificial neural networks, the speaker provides an example calculation. They multiply the values of x1, zero, and -1.5 by their respective weights. The resulting values are 0.5, 0, and -1.5, respectively. The sum of these values is negative 1, which is the activation. However, since the activation is below the threshold value of zero, the output should be zero.

1.1.50 How Powerful is a Perceptron Unit

Perceptron units are powerful in that they can compute linear functions to classify inputs into two categories (0 or 1). The perceptron's behavior is determined by the weights assigned to the inputs and a threshold value. By setting specific weights and threshold, we can create a line (or hyperplane) in the input space that separates inputs into two regions corresponding to the output values. The perceptron always computes linear functions and divides the input space into two halves, with points above the line representing 1s and points below the line representing 0s. Therefore, perceptrons are limited to computing linear functions and cannot represent more complex relationships.

1.1.51 How Powerful is a Perceptron Unit Quiz Question

This lecture addresses the power and function of a perceptron unit in the context of decision trees. The speaker presents an example of a random function and focuses on the case where X1 and X2 are limited to the values zero and one. The lecture introduces a quiz question to determine the relationship and function being computed by Y, emphasizing that there is a one-word answer that can be deduced by analyzing the input-output mapping.

1.1.52 How Powerful is a Perceptron Unit Quiz Solution

The lecture discusses binary functions and boolean functions, where 0 represents false and 1 represents true. It explores the concept of conjunction (AND) in machine learning and how it can be represented in a binary space. The lecturer poses a question about whether it is possible to represent disjunction (OR) in the same space and suggests a quiz to explore this idea further.

1.1.53 How Powerful is a Perceptron Unit OR Quiz Question

This lecture discusses the task of creating a perceptron unit that performs the "or" function. The goal is to find appropriate values for weight one, weight two, and theta in order to achieve the desired output. It is emphasized that there are multiple valid solutions for this problem.

1.1.54 How Powerful is a Perceptron Unit OR Quiz Solution

In this lecture on decision trees, the instructor discusses the concept of moving a line to classify data points. The objective is to classify three points in the green zone as 1 and the remaining point in the red zone as 0. To achieve this, a threshold and weights are needed, with either X1 or X2 being sufficient to cross the line. The lecturer explains two possible approaches: setting the weight of X1 to 1 and the threshold to 1, or keeping the weights as they were and adjusting the threshold to a lower value. Both methods successfully classify the points. The lecture concludes with the consideration of the "not" operation on one variable.

1.1.55 How Powerful is a Perceptron Unit NOT Quiz Question

In this lecture on decision trees, the instructor discusses the concept of a perceptron unit and its implementation in a not gate. The perceptron unit takes on different values for a variable X1. For the not gate, when X1 is zero, the output should be one, and when X1 is one, the output should be zero. The instructor asks the students to determine the appropriate weight and threshold (theta) values to achieve this behavior.

1.1.56 How Powerful is a Perceptron Unit NOT Quiz Solution

To represent the Boolean functions AND, OR, and NOT using perceptron units, the weight can be set to -1 and the threshold to 0. This effectively flips the values of 0 and 1, with the weight changing a 0 to 0 and a 1 to -1. This creates a dividing line at 0, making anything less than or equal to 0 fall under the threshold. By combining these perceptron units, any Boolean function can be represented. The lecture mentions that XOR, also known as parity, is a particularly challenging function to represent using a single perceptron unit.

1.1.57 XOR as Perceptron Network Question

The lecture discusses how to represent XOR (exclusive OR) operation using a network of perceptrons, which is not possible with a single perceptron. The goal is to compute XOR using two units: one that computes the AND operation and another that computes the OR operation. The lecture asks students to figure out how to set the weights and threshold for the second unit, which has three inputs (X1, X2, and the AND of X1 and X2), in order to compute XOR.

1.1.58 XOR as Perceptron Network Solution

To solve the XOR problem using a perceptron network, we can use a combination of AND, OR, and NOT functions. By analyzing the truth tables for AND, OR, and XOR, we can deduce that the last node in the network should compute the OR of X1 and X2, except in the case of the last row where it should turn off when AND is true. This node effectively computes (OR - AND). To achieve this, we can assign weights of one to the two inputs and a threshold of one, which will make the unit turn on if either X1 or X2 are on. By introducing a negative weight of -1 to subtract the output of the AND function, we can get the desired result in most cases. However, this approach still has a problem when both X1 and X2 are on, as the sum will be two, and subtracting one will still result in a positive value. To address this, we can try subtracting two instead. This will give us a sum of zero, which is less than the threshold, so the node will output zero. In the other two cases, when the AND function is off, the node will act like the OR function. In summary,

we can create a perceptron network solution for XOR by using the formula (OR - AND) and adjusting the weights accordingly. There are many possible solutions to this problem.

1.1.59 Perceptron Training

In this lecture, two different rules for setting weights in machine learning are discussed: the Perceptron Rule and gradient descent or the Delta Rule. The Perceptron Rule utilizes threshold outputs, while the Delta Rule uses unthreshold values. The lecture focuses on the Perceptron Rule and how to set weights for a single unit to match a training set. The lecture introduces the concept of a bias unit and its corresponding weight, which is treated as the negative threshold value. The lecture then explains the process of updating the weights based on the desired output and the actual output of the network. The lecture concludes by discussing the capability of the Perceptron Rule to find a line that separates positive and negative examples in a linearly separable dataset. However, the lecture also addresses the difficulty of determining when a dataset is linearly separable and the challenge of knowing when to stop the algorithm.

1.1.60 Gradient Descent

In this lecture, we explore the concept of Gradient Descent as an algorithm that can handle non-linear separability. We review the previous approach of using a summation over input features and weights to determine the activation, which is then compared to a threshold for classification. Instead of thresholding during training, we aim to minimize the error between the activation and the target value by adjusting the weights. This error metric is defined as the sum of half the squared difference between the target and activation for each example in the dataset. We introduce calculus to determine how changes in weights affect the error, and calculate the partial derivative of the error metric with respect to each weight. The chain rule is used for these partial derivatives, simplifying the calculation due to the presence of half in the error metric. Ultimately, we find that the update for each weight is the product of the difference between activation and target output with the activation on the corresponding input unit. This update rule resembles the one used in the perceptron algorithm, but with the additional factor of the learning rate.

1.1.61 Comparison of Learning Rules

The lecture discusses two update rules for learning: the gradient descent rule and the perceptron rule. The gradient descent rule involves moving the weights in the negative direction of the gradient, while the perceptron rule involves thresholding the activation and using it as an input. These rules result in two different algorithms with different behaviors. The perceptron algorithm has finite convergence, but only for linearly separable data. The gradient descent rule is more robust to non-linearly separable data but only converges to a local optimum.

1.1.62 Comparison of Learning Rules Quiz Question

The lecture discusses the question of why a gradient descent type algorithm is not used with an error metric defined in terms of the desired output (y hat) instead of the activation (a). Several possible explanations are explored, including computational infeasibility, non-differentiability of the current form, instability due to weights growing too fast, and the potential for multiple different answers. Ultimately, the goal is to prevent the problem from being ill-defined.

1.1.63 Comparison of Learning Rules Quiz Solution

The main reason for not using gradient descent on y hat is that it is not differentiable due to being a discontinuous function. The activation function has a step function jump at zero, making it difficult to take the derivative. This lack of differentiability makes it impractical to use this algorithm for fixing weights. However, if the function were made more differentiable by smoothing out the discontinuity, it could potentially be used.

1.1.64 Sigmoid

The lecture discusses the sigmoid function, which is defined as the function of the activation value. The sigmoid function is computed as one over one plus e to the power of minus a. The function approaches zero as

the activation approaches negative infinity and approaches one as the activation approaches positive infinity. The sigmoid function has a gradual transition between negative five and five, making it differentiable. The derivative of the sigmoid function is equal to the function itself multiplied by one minus the function itself. The derivative flattens out for very large and very negative activation values. When the activation is zero, the sigmoid function evaluates to one-half, and the derivative at that point is one-fourth. The sigmoid function is in a nice form for working with thresholds, but other functions with different properties could also be used.

1.1.65 Neural Network Sketch

Neural networks are constructed using sigmoid units to create a chain of relationships between the input layer and the output. Hidden layers compute the weighted sum, which is then sigmoided. This mapping from input to output is differentiable in terms of the weights, allowing for weight adjustments to produce the desired output. Backpropagation is a computationally beneficial organization of the chain rule, allowing for the computation of derivatives and weight updates. If the sigmoid units are replaced with a differentiable function, the same weight adjustment can still be made. However, neural networks do not act exactly like perceptrons and may encounter local optima in the error function. The error function can have multiple low points, but only one is the global minimum.

1.1.66 Optimizing Weights

When using gradient descent to optimize weights in complex neural networks, there is a possibility of getting stuck in local minima. To address this, advanced optimization methods can be used. One such method is using momentum terms in the gradient, which allows for continuance in the direction of descent to avoid getting stuck in a local minima. Higher order derivatives can also be utilized to optimize weight combinations. Randomized optimization techniques can increase robustness. Additionally, there may be a need to penalize complex networks by limiting the number of nodes or layers, as well as keeping weight values within a reasonable range to mitigate overfitting.

1.1.67 Restriction Bias

The lecture discusses the restriction bias and inductive bias of neural networks. The restriction bias refers to the representational power and set of hypotheses that the network of neurons can consider. Neural nets are not very restrictive and can represent a wide range of functions as long as they have a complex enough network structure. They can represent both boolean and continuous functions, even if they are discontinuous, by adding more hidden layers. However, this flexibility also raises the concern of overfitting. To address this, techniques like cross-validation can be used to determine the appropriate number of hidden layers and nodes, as well as when to stop training if the weights become too large. Neural network training is an iterative process where the error on the training set decreases, but the error on a separate test set or cross-validation set can first decrease and then increase, indicating the point at which training should be stopped. This turnaround point is often associated with the weights becoming larger. By understanding this difference and employing appropriate techniques, neural networks can be effectively trained without overfitting.

1.1.68 Preference Bias

Preference bias refers to the algorithm's tendency to prefer one representation over another. In the case of decision trees, preference bias is influenced by criteria such as high information gain, shorter tree length, and accuracy. When initializing weights in machine learning algorithms like neural networks, it is common to use small random values to avoid local minimums and overfitting. Starting with small values also indicates a preference for simpler explanations over complex ones. This preference aligns with Occam's razor, which states that entities should not be multiplied unnecessarily. In the context of supervised learning, choosing simpler hypotheses has been shown to result in better generalization error.

1.2 Instance Based Learning

1.1.69 Summary

In this lecture on neural nets, the speaker concludes the discussion on the topic and mentions that there will be additional material covered in the homework. They then ask Charles to recall what was covered in the neural net section. Charles mentions that they learned about perceptrons, which are linear threshold units that can be combined to produce any Boolean function. They also learned about a learning rule for perceptrons, which works for linearly separable data sets. Additionally, they learned about a general differentiable rule using gradient descent for propagation, and briefly discussed the preference and restriction of neural networks. The lecture ends with the speaker saying goodbye and mentioning that they will meet again next time.

1.2.1 Instance Based Learning Before

In this lecture, the instructor introduces the concept of instance-based learning as a different approach compared to previous learning algorithms discussed. The lecture aims to uncover the unspoken assumptions made so far. The instructor begins by explaining the process of supervised learning, where training data is used to learn a function that represents the data. However, the data is then discarded and future data points are evaluated using the learned function without reference to the original data. The instructor proposes an alternative approach that does not discard the data.

1.2.2 Instance Based Learning Now

The lecture introduces the concept of instance-based learning, which involves storing all training data in a database and retrieving information from the database when new data is encountered. This approach eliminates the need for complex learning algorithms and allows for reliable and fast processing. However, there are limitations, such as the lack of generalization and sensitivity to noise. The lecture highlights the need to address these limitations by finding ways to overcome the literal interpretation of storing and retrieving data.

1.2.3 Cost of the House

The lecture discusses the use of instance-based learning in machine learning. The instructor presents a graph showing houses labeled with different colors based on their price range. The goal is to use machine learning techniques to predict the price range for new houses represented by black dots. The instructor suggests using the nearest neighbor approach to make predictions based on the closest labeled houses. This approach works well for most cases, but not for houses that are located near houses with conflicting labels. To address this issue, the instructor suggests considering a larger context and looking at more neighbors instead of just the closest one.

1.2.4 Cost of the House Two

The lecture discusses the idea of instance-based learning using the example of predicting the color of a house in a neighborhood. The lecturer suggests using the concept of distance to determine the nearest neighbors and their colors as a way to make predictions. The lecture highlights the importance of defining distance and considers different measures such as straight-line distance, driving distance, or Google Maps distance. It is mentioned that distance is a standard for measuring similarity. The algorithm, named "k nearest neighbors," is introduced, where 'k' represents the number of neighbors considered. The lecture also mentions the possibility of adding additional features or dimensions, such as square footage, veterans, school district, to improve similarity calculations. The lecturer concludes by stating that the algorithm addresses the issues of overfitting and handling missing data. The lecture suggests turning the discussed ideas into a formal algorithm.

1.2.5 K NN

The K-NN algorithm is a simple instance-based learning algorithm that uses a distance metric and a number of neighbors to classify or regress a new query point. In the classification case, the algorithm takes a vote

of the labels (yi values) of the nearest neighbors, with ties broken by choosing the most frequent label. In the regression case, the algorithm takes the mean of the nearest neighbors' labels as the output. If there are ties in the distances, all points that are closest are considered. The algorithm can be customized by choosing different distance metrics, handling ties differently, or using weighted voting or averaging based on similarity. The simplicity of the algorithm comes from leaving many decisions up to the designer, which can significantly impact the results.

1.2.6 Wont You Compute My Neighbors Question

The lecture discusses three learning algorithms: one neural net, one nearest neighbor, K nearest neighbor, and linear regression. The instructor asks the student to fill in a table with the running time and space requirements for each algorithm. The running time refers to the time it takes to learn and query, while space refers to the space needed for learning and querying. The data points are assumed to be sorted in a one-dimensional space. For one nearest neighbor, the running time of learning is constant as there is no learning involved. The space requirements are big O of N, where N is the number of data points. The student is then asked to fill in the remaining information.

1.2.7 Wont You Compute My Neighbors Solution

In the lecture on nearest neighbor and linear regression in the context of instance-based learning, it is explained that one nearest neighbor can be found using binary search in sorted data, with log time complexity. If the data is unsorted, it would require linear time complexity. Both nearest neighbor and linear regression require a small amount of space, with nearest neighbor requiring constant space and linear regression requiring two constants. In terms of time complexity, nearest neighbor has logarithmic complexity for querying, while linear regression has linear complexity. The trade-off between learning and querying is discussed, with nearest neighbor being a lazy learner that puts off learning until necessary, while linear regression is an eager learner that learns immediately. The importance of balancing the workload between learning and querying is highlighted, depending on the frequency of querying and the trade-off desired.

1.2.8 Domain K NNowledge Question

In this lecture, the instructor discusses the k-nearest neighbors (k-NN) algorithm and how different choices in distance metrics and values of k can produce different results. The training data for this regression problem consists of xy pairs, with x being two-dimensional. The goal is to produce the proper output for a given query point. Four different cases are considered: using Euclidean distance with 1 nearest neighbor, using Euclidean distance with 3 nearest neighbors and averaging their outputs, using Manhattan distance with 1 nearest neighbor, and using Manhattan distance with 3 nearest neighbors and averaging their outputs. In cases of ties, the college ranking trick of including everyone who is at least as good as the k closest is used. The instructor encourages students to ask questions, but the student feels ready to tackle the problem.

1.2.9 Domain K NNowledge Solution

In this lecture, the instructor and a student named Michael discuss the concepts of Euclidean distance and Manhattan distance in the context of instance-based learning. They compute these distances for a set of data points and discuss the nearest neighbor approach for finding the closest points. They also calculate the average of the output values for the nearest neighbors. They then compare the results for Euclidean distance and Manhattan distance and note that they produce different answers. The instructor explains that the choice of distance metric can greatly impact the results. They also introduce a specific function, $Y = X1^2 + X2$, and show that none of the computed distances match the true value of the function. The instructor concludes that while k-Nearest Neighbors (kNN) is a simple and effective algorithm, it may not always yield accurate results depending on the assumptions and biases inherent in the data.

1.2.10 K NN Bias

Preference bias refers to the notion of why we prefer one hypothesis over another. In the case of kNN (k-nearest neighbors), it has a built-in preference bias. There are three biases related to kNN: locality, smoothness, and importance of features. Locality means that near points are similar to each other. Smoothness assumes that functions behave smoothly, with similar points being more related than dissimilar points.

The importance of features refers to the assumption that all features matter equally. However, different features may have different levels of importance. For example, in the Euclidean or Manhattan distance, if one feature is squared while the other is not, the squared feature becomes more important. Taking these biases into account is important in making assumptions about the domain knowledge. Additionally, the relevance of features is an important factor, and kNN may have a weakness in this aspect. It is relevant to mention a fundamental result of machine learning that highlights the importance of relevance in kNN and other algorithms.

1.2.11 Curse of Dimensionality

The curse of dimensionality refers to the exponential increase in the amount of data needed to accurately generalize as the number of features or dimensions in a dataset grows. This poses a problem for machine learning because the instinct to add more features to understand the data better leads to exponentially increasing data requirements. Each additional feature adds another dimension to the input space, requiring exponentially more data to generalize accurately. This challenge highlights the difficulty in determining which features are relevant and important in a dataset, as a similarity function that assumes all features are equally important cannot separate irrelevant features without a significant amount of data.

1.2.12 Curse of Dimensionality Two

The curse of dimensionality refers to the exponential increase in the number of data points needed to represent a given space as the number of dimensions increases. This has implications for instance-based learning methods like k-nearest neighbors (kNN) where points are used to represent or cover the space. For each additional dimension, the number of points required to maintain the same coverage grows exponentially. This is a problem for machine learning in general, as it requires a larger training set to fill the increased space. Adding more dimensions to a machine learning problem without increasing the amount of data can lead to sparsity and poor performance. Strategies for dealing with the curse of dimensionality will be covered in future lessons.

1.2.13 Some Other Stuff

In this lecture on instance-based learning, the instructor discusses some additional topics related to the curse of dimensionality. The lecture focuses on the assumptions made about parameters in the algorithm, specifically the distance measure used. The two main distance measures discussed are Euclidean and Manhattan, which are useful for regression problems involving numbers. However, for discrete data or cases where similarity is determined based on specific features, other distance functions can be used. The lecture also touches on the selection of the value for k, the number of neighbors to consider. When k equals the total number of data points (n), a simple average would result in a constant function. However, a weighted average can yield different results depending on the position of the query point. The concept of locally weighted regression is introduced, where instead of using a simple average, different regression models can be used to estimate the target variable based on the nearby points. This allows for more powerful and complex hypotheses to be formed. The lecture concludes by highlighting the ability of kNN to leverage local information to build concepts and create arbitrarily complicated functions.

1.3 Ensemble B & B

1.2.14 What Have We Learned

In this lecture, the instructor recaps key concepts discussed in the lesson on computational learning theory. The focus of the lesson was on understanding what is learnable and the sample complexity of learning algorithms. Sample complexity refers to the number of examples needed to learn a concept. The instructor highlights that data is an important resource in machine learning and discusses different scenarios in the teacher-student relationship that can affect sample complexity. The scenarios include the learner asking all the questions, the teacher picking the questions, and the teacher being nature (fixed distribution). The lecture also covers mistake bounds, which measure the number of mistakes made during learning, and introduces the concept of version spaces and PAC (Probably Approximately Correct) learnability. The instructor emphasizes the distinction between training error, test error, and true error, which is connected to

the distribution of data. The lecture concludes with the introduction of a sample complexity bound formula, which depends on the size of the hypothesis space, target error bound, and failure probability. The instructor addresses two questions raised by a student regarding target concepts not present in the hypothesis space and infinite hypothesis spaces. The discussion about infinite hypothesis spaces is postponed to a future lesson.

1.3.1 Ensemble Learning Boosting

This lecture is about ensemble learning, specifically focused on boosting. The instructor begins by discussing the problem of classifying spam emails and proposes using simple rules as indicators of spam instead of complicated algorithms like decision trees or neural networks. Examples of these rules include the presence of certain words, messages from certain senders, message length, and image-only emails. The instructor explains that while each rule provides some evidence, none of them alone are sufficient to accurately determine if a message is spam. This is where ensemble learning comes in. Ensemble learning combines multiple simple rules to make a more accurate decision. The instructor mentions that decision trees and neural networks can also be seen as ensembles of simpler rules.

1.3.2 Ensemble Learning Simple Rules

Ensemble learning is a technique in machine learning that combines multiple simple rules, which individually may not provide accurate results, to create a more complex rule that performs well. The basic form of ensemble learning involves learning over subsets of data to generate different rules, which are then combined into a single complex rule. By analyzing different subsets of data, rules can be discovered that are effective for specific subsets, but may not perform well on the entire dataset. After collecting these rules, they are combined to form the final rule. In the context of classifying emails, for example, one subset of emails may show that the word "manly" is a common characteristic of spam emails, while another subset may reveal that spam emails are often short or contain only images or URLs. By combining these rules, a more accurate classification rule can be created. Ensemble learning is effective because analyzing subsets of data allows for the identification of simple rules that would be difficult to find when considering the entire dataset.

1.3.3 Ensemble Learning Algorithm

Ensemble Learning Algorithm: - Ensemble Learning algorithm involves picking a subset of data, applying a learning algorithm to it, and then combining the resulting rules. - The combination process is important and should be done intelligently. One simple way is to uniformly and randomly choose subsets of data, apply a learning algorithm, and then combine the results. - In the case of regression, a simple way to combine the rules is by averaging them, taking the mean. - Reasons for not equally believing in each rule could be due to bad random subsets or certain subsets having more error or using more complex rules. - Exploring how this combination process can go wrong can be done through quizzes.

1.3.4 Ensemble Learning Outputs Question

In this lecture, the topic of discussion is ensemble learning outputs. The scenario presented involves having N data points and using a zeroth order polynomial as the learner for subsets of the data. The output of the learners will be combined by averaging them. The subsets are constructed randomly and each subset contains only one data point. The lecturer asks what the ensemble output will be and requests a short description or a number as the answer.

1.3.5 Ensemble Learning Outputs Solution

In this lecture, the speaker discusses ensemble learning outputs and the solution for combining the outputs of individual learners. The ensemble rule for this scenario is to learn over subsets using a zeroth order polynomial. The expected error is minimized by taking the average output value of each individual point. The combining algorithm then combines these outputs using the mean. The ensemble outputs can be referred to as average, mean, zeroth order polynomial, one node decision tree, or a constant equal to the mean of the data. This approach is similar to unweighted average with kNN, where k is equal to n. However, the

speaker suggests that a smarter approach should be considered. The speaker proposes examining housing data and finding a better solution.

1.3.6 Ensemble Learning An Example

In this lecture, the instructor discusses ensemble learning using an example with housing data. They begin by explaining cross-validation and how the training and test sets are combined on the same slide. The instructor then randomly selects subsets of five data points and learns third order polynomials on each subset. These polynomials are combined by averaging, creating a red line plot that closely matches a fourth order polynomial (blue line) learned using simple regression. The instructor notes that while the blue line performs better on the training set, the red line performs better on the test set. The discussion centers around the concept of overfitting and how ensemble learning, specifically bagging, can mitigate this issue by averaging out variances and differences. Bagging, also known as bootstrap aggregation, involves randomly selecting subsets of data and combining them by the mean. The lecture concludes by noting that while bagging is effective, there may be room for improvement and introduces the topic of boosting, which will be discussed in the next lecture.

1.3.7 Ensemble Boosting

In this lecture, the speaker discusses the concept of boosting as an alternative to randomly selecting subsets of data. The idea is to focus on examples that are difficult to classify, rather than uniformly random subsets. The difficulty of an example can be relative to the current level of learning or in an absolute sense. Boosting aims to improve performance by focusing on examples that are not yet mastered. The combining of these rules is done through a weighted mean, which avoids the limitations of simply averaging the rules. However, the speaker acknowledges the challenge of not losing track of previously mastered examples, which will be addressed by the particular weighting approach. Two technical definitions will be introduced to further explain this concept.

1.3.8 Ensemble Boosting Quiz Question

The lecture discusses the concept of error in machine learning and introduces a new definition of error. Error is typically defined as the square difference between the correct labels and the output produced by a classifier or regression algorithm for regression tasks, or the number of mismatches for classification tasks. However, this definition assumes that every example is equally important, which is not always the case. The lecturer introduces a new definition of error, denoted as subscript D, which takes into account the distribution of examples. It is defined as the probability that the learner will disagree with the true concept on a particular instance X, given the underlying distribution. The lecturer provides an example to illustrate the difference between the new definition and the number of mismatches, asking the students to determine the error rate when a learner's hypothesis correctly predicts the first and third instances but incorrectly predicts the second and fourth instances.

1.3.9 Ensemble Boosting Quiz Solution

In this lecture, Michael is asked to answer a question about mismatches in a set of examples. The speaker explains that in a scenario where the examples are equally likely to be seen, the number of mismatches is half. However, the speaker then introduces a different scenario where each example is likely to be seen in different proportions. Specifically, the first example is seen half the time, the second example is seen one 20th of the time, the fourth example is seen one 20th of the time, and the third example is seen four tenths of the time. Michael understands this distribution of probabilities.

1.3.10 Ensemble Boosting Quiz Two Question

The speaker poses a question to the audience about the error rate.

1.3.11 Ensemble Boosting Quiz Two Solution

In this lecture on Ensemble Boosting, the speaker emphasizes the importance of considering the underlying distribution of examples when measuring error in machine learning. They introduce the concept of a

weak learner, which is a learning algorithm that performs better than chance regardless of the data distribution. The expected error of a weak learner is always less than half, as expressed by the inequality $P(disagreement) \leq 1/2 - \epsilon$, where ϵ represents a small number close to zero. The speaker also mentions that boosting techniques use this distribution to define which examples are more important to learn. The lecture concludes with a suggestion to conduct a quiz to further clarify concepts.

1.3.12 Weak Learning Question

The lecturer introduces the concept of weak learning and presents a quiz to test understanding. The quiz involves a matrix with three hypotheses and four examples. The lecturer explains that no hypothesis gets everything right, and asks the student to come up with a distribution over the examples where a learning algorithm can choose a hypothesis that performs better than chance. The student is also asked to determine if there is a distribution where the learning algorithm cannot choose a hypothesis with an expected error greater than half. The lecturer emphasizes the importance of considering the distribution when determining expected error. The student is encouraged to fill in zeroes if no such distribution exists and to ensure that all values in each column add up to one.

1.3.13 Weak Learning Solution

In this lecture, the speaker discusses the concept of weak learning solutions in the context of a specific example. The speaker explores different weight distributions on four examples (X1, X2, X3, and X4) and evaluates the performance of three hypotheses (H1, H2, and H3) based on these weight distributions. The first attempt at equal weights on all four examples yields H1 with a 75% correct rate. However, giving all the weight to X1 results in H1 and H2 having 100% error rates, while H3 performs perfectly. This leads to the realization that there may not be an evil distribution, as all examples have a hypothesis that performs well. By experimenting with different weight distributions, it is determined that the correct answer is 1/2, 1/2, 0, 0. The speaker concludes that there is no weak learner for this hypothesis space on this example set, but notes that with more hypotheses and examples, it would be possible to find weak learners for all distributions. The speaker also explains that having many hypotheses that perform well on different examples is crucial for finding weak learners, emphasizing the importance and strength of this condition.

1.3.14 Boosting In Code

Boosting is a technique used in binary classification tasks, where the goal is to find a weak classifier that performs well on the training set. The boosting algorithm works by constructing a distribution over the training examples at each time step and finding a weak classifier with a small error. The weak classifier is a hypothesis that outputs a prediction for each example. The error of the weak classifier is defined as the probability of it disagreeing with the training label, with respect to the underlying distribution. The boosting algorithm repeats this process for a number of time steps, constantly updating the distribution and finding new weak classifiers. Finally, the algorithm combines all the weak classifiers into a final hypothesis. The details of how the distribution and final hypothesis are obtained are not provided in this lecture.

1.3.15 The Most Important Parts

The lecture begins by discussing the base case of the distribution at the beginning of time, denoted as D1. The lecturer sets the distribution to be uniform, meaning each example is equally likely. The motivation for this is that there is no reason to believe any example is more important or harder than another. Next, the lecturer explains how to construct the distribution at each time step T. The new distribution (D T+1) is calculated as the old distribution (D T) multiplied by e to the power of -alpha T times the label (Y sub i) times the hypothesis (H sub T) of the example (X sub i), all divided by a normalization factor (Z sub T). The lecturer explains the meaning of each term in the calculation: D is the distribution, representing the importance or frequency of an example. H T returns either -1 or 1, as the training set labels are -1 or 1. Y sub i is the label for the example X sub i, and alpha T is a positive constant. If the hypothesis at time T agrees with the label for an example, the product of Y sub i and H sub T is 1, resulting in the raising of e to a negative number. If they disagree, the product is -1, resulting in the raising of e to a positive number. Playing around with the numbers shows that the error is always between 0 and 1, and the natural log of 1 minus a number between 0 and 1 divided by that number is always positive. In conclusion, when the

hypothesis and label agree, the relative weight of D sub T of i is decreased, and when they disagree, the weight is increased.

1.3.16 When D agrees Question

In this lecture, the professor introduces a quiz question about the distribution over a specific example when the hypothesis agrees with the label. There are four possible outcomes when they agree: the probability of seeing that example increases, decreases, or stays the same depending on the old value of d and alpha. Students are instructed to choose the correct answer from the provided options.

1.3.17 When D agrees Solution

In this lecture on ensemble learning, the speaker discusses the behavior of the algorithm when two classifiers agree or disagree. When the classifiers agree, the algorithm scales down the output, resulting in a value between zero and one. The normalization constant, x sub t, remains constant throughout the process. If one classifier is correct and the other is incorrect, the algorithm decreases the weight of the correct classifier and increases the weight of the incorrect classifier. Conversely, when the classifiers disagree, the algorithm puts more weight on the incorrect classifier, assuming it is harder to classify. It is important to note that the behavior of the algorithm depends on the specific situation and normalization process. Overall, the algorithm aims to improve the classification performance by focusing on the classifiers that were initially getting wrong answers.

1.3.18 Final Hypothesis

The final hypothesis in ensemble learning is constructed by taking a weighted average of the weak classifiers. The weights are determined by alpha sub T, which is a measure of the classifier's performance relative to the overall error. The final hypothesis is obtained by passing the weighted sum through a thresholding function, where positive values are classified as positive, negative values as negative, and zero values as zero. This algorithm is called boosting and involves constructing the distribution and combining the weak classifiers. The rationale for using natural logarithms and alpha sub T is explained in the reading material.

1.3.19 Three Little Boxes

The lecture introduces the concept of boosting, a machine learning algorithm that combines multiple weak learners to create a strong learner. The example used is a classification problem with red pluses and green minuses in a square region of a plane. The hypothesis space is defined as axis-aligned semi-planes, where a line separates the positive and negative regions. Boosting is applied to this example, with the first learner choosing a vertical line to separate the first two data points from the rest. This hypothesis has an error of 0.3 and an alpha of 4.2. The second learner adjusts the distribution to give more weight to the misclassified examples, making the three green minuses more prominent. The learner's hypothesis is a line to the right of the three pluses, correctly classifying them but misclassifying three minuses. This hypothesis has an error of 0.21 and an alpha of 0.65. The distribution is adjusted again for the third learner, making the three green minuses even more prominent and making the pluses smaller. The two red pluses on the left, never misclassified, become less prominent. The two misclassified minuses become smaller as well. The third hypothesis is not revealed in the lecture.

1.3.20 Which Hypothesis Question

The lecture involves a quiz on selecting the correct hypothesis. Three possibilities (A, B, and C) are presented, each represented by a line. The choices are a horizontal line (A), another horizontal line (B), or a vertical line (C). The correct answer is unclear.

1.3.21 Which Hypothesis Solution

The lecture discusses the selection of a hypothesis solution and its application in an ensemble learning system. The speaker evaluates three hypotheses, labeled A, B, and C, and ultimately chooses hypothesis A as the best option for separating data points. The speaker also highlights the weights and errors associated with each hypothesis, showing a decreasing error over time and a higher weight for hypothesis A. The

speaker then demonstrates how combining the weighted hypotheses produces a more complex overall hypothesis that accurately captures positive and negative examples. This approach is compared to decision trees, neural networks, and weighted nearest neighbor methods. The speaker concludes by noting that the combination of simple hypotheses through weighted averages allows for increased expressiveness and complexity in the final hypothesis. The use of sums in this combination is surprising, but the application of a non-linearity, specifically the sine function, helps create a nonlinear effect in the overall hypothesis.

1.3.22 Good Answers

Boosting is a machine learning technique that re-weights examples to focus on misclassified ones. The intuition for why boosting works well is that there cannot be many examples that are misclassified, resulting in the error rate decreasing over time. Each iteration of boosting forces the learner to do better on the difficult examples. The error rate may not necessarily improve with each iteration, but it should not get worse. When examples are continuously misclassified, their importance decreases, and the learner has to focus on getting them correct, as well as other examples. Cycling between examples becomes difficult due to the exponentially increasing weight on misclassified examples. Information gain is a key aspect of boosting, in which the algorithm continuously picks up new information to improve its performance. The final hypothesis obtained through boosting must not only perform well on the last set of difficult examples but also on a variety of data.

1.4 Kernel Methods and SVMs

1.3.23 Summary

In this lecture on ensemble learning, the instructor discusses the concept of learning multiple times in multiple ways. Bagging is introduced as a simple ensemble technique, where multiple classifiers are trained on different subsets of the training data and their predictions are merged together. The benefits of ensembles and bagging are discussed, including the ability to combine simple classifiers to obtain more complex ones. The lecturer also introduces the idea of boosting, which can improve the performance of weak learners by iteratively training them on weighted versions of the training data. Boosting is highlighted as a powerful technique that can significantly reduce errors. The lecture emphasizes that boosting is agnostic to the type of learner used, as long as it is weak. The concept of weak learners is briefly explained, as well as the importance of understanding the meaning of error in relation to an underlying distribution. The lecturer also mentions that boosting has the advantage of being fast and provides an explanation for the phenomenon of improving test error even as training error improves over time. The lecture concludes with the promise of further explanation in the next session.

1.4.1 The Best Line Question

In this lecture, the focus is on support vector machines (SVMs) and their relationship to boosting. The lecturer starts by introducing a quiz to determine the best line for separating two classes of labeled points. The challenge is to select one green dot and one red dot to define the line, without any predefined criteria for what makes a line the best. The lecturer encourages Michael to justify his selection, even if he believes all options are equally good.

1.4.2 The Best Line Solution

In this lecture, the speaker discusses the concept of finding the best line for separating data points. They consider nine different lines and conclude that the middle line is the best choice. They argue that lines too close to the positive or negative data points may overfit the data, and it is important to not believe the data too much. Overfitting can lead to problems when new data is encountered. The middle line is seen as the line that is consistent with the data while committing the least to it. This concept of finding the line of least commitment is the basis behind support vector machines. The speaker then suggests finding an equation to define such a line, as it may be difficult to do so visually with higher-dimensional data points.

1.4.3 Support Vector Machine

In this lecture, the instructor discusses support vector machines (SVMs) and the concept of finding a decision boundary that maximizes the distance between the boundary and the data points. The instructor introduces the notation for hyperplanes in multi-dimensional space and explains how the output of the classifier determines whether a point belongs to the positive or negative class. The equation of the decision boundary is defined as w transpose x + b = 0, where w represents the parameters of the plane and b moves it out of the origin. The instructor then discusses the equations for the lines that brush up against the positive and negative examples, which are w transpose x + b = 1 and w transpose x + b = -1, respectively. The goal is to maximize the distance between these two lines, which represents the distance between the decision boundary and the data points. This distance is determined by the difference between the equations of the positive line and the negative line. The instructor concludes the lecture by subtracting these equations to obtain a single equation representing the distance between the lines.

1.4.4 Distance Between Planes Question

In this lecture, the instructor presents a quiz involving two equations for hydroplanes that are parallel to each other. The goal is to subtract the two equations and solve for the line described by their difference. The instructor emphasizes that the output should be the distances between the two planes, represented by X1 and X2. The student, Michael, understands the task and plans to subtract the second equation from the first equation to solve it.

1.4.5 Distance Between Planes Solution

The lecture discusses the difference between two equations and how to determine the distance between two points, x1 and x2. The lecturer emphasizes the importance of expressing the distance in terms of the vector W, which defines the line. The lecturer introduces the idea of dividing both sides of the equation by the length of W to eliminate it on one side, resulting in x1 minus x2 being projected onto the unit sphere. The lecturer explains that the length of x1 minus x2 in the W direction is equivalent to the distance between the two points. The lecturer then connects this concept to the parameter W, which is perpendicular to the line and represents the line. The goal is to find the parameters that maximize the distance while correctly classifying the data points. This concept is called the margin, and finding the optimal decision boundary is equivalent to maximizing the margin. The lecture concludes by stating that this understanding can be used to solve the problem of finding the best line.

1.4.6 Still Support Vector Machines

Support Vector Machines (SVMs) aim to maximize the equation $2/W$, subject to the constraint of accurate classification. This constraint can be expressed as $YI(W^T * XI + B) \geq 1$ for all training data examples. This can be transformed into the equivalent problem of minimizing $1/2 * W^2$. Solving this quadratic programming problem is easier than the original problem and techniques from linear algebra can be used. The problem can be further transformed into the normal form for a quadratic programming problem, where we maximize a function with parameters alpha. This function involves the sum of alpha values multiplied by labels and data values, subject to constraints. These constraints include non-negativity of alpha values and the sum of the product of alpha and labels equals zero. Solving this quadratic programming problem will provide the optimal solution for SVMs.

1.4.7 Still More Support Vector Machines

The lecture discusses the properties of the solution for a quadratic programming equation used to maximize a certain equation. It is mentioned that once the alphas that maximize the equation are found, the W (not to be confused with the big W) can be recovered easily. The value of B can also be recovered using this information. It is pointed out that most of the alphas in the solution are usually zero, indicating that only a few data points are important for finding the optimal W. These data points, known as support vectors, provide all the support for W. It is stated that only a few support vectors are needed to build a machine that can find the optimal W.

1.4.8 Optimal Separator Question

In this lecture, the speaker discusses the concept of an optimal separator. They present a graph with positive and negative points and a green line representing the optimal separator. They mention that some alphas will be zero and some will not. The speaker asks the listener to identify a positive example and a negative example that are likely to have zero alphas and are not part of the support vectors. The listener is confused about whether they should choose examples with zero or non-zero alphas, but ultimately selects examples with zero alphas.

1.4.9 Optimal Separator Solution

The lecture discusses the concept of optimal separator solution in the context of kernel methods and Support Vector Machines (SVMs). It emphasizes that points close to the decision boundary have a stronger influence than points that are far away. The lecturer suggests that points that are far away from the decision boundary do not matter, regardless of their label (plus or minus). This concept is compared to nearest neighbors and described as a way of improving instance-based learning by determining which points to keep and which to discard. The lecture also introduces the importance of the dot product (xi transpose xj) in the equation. The dot product represents the projection of one vector onto another, indicating the similarity and direction of the vectors. It is considered a measure of similarity between points. The lecturer explains that the equation in SVMs finds pairs of points that matter for defining the decision boundary and examines their similarity in terms of their output labels.

1.4.10 Linearly Married

In this lecture, the professor discusses a problem in linear separation of points using support vector machines (SVMs). Initially, the professor presents a graph with two clouds of points and demonstrates a linear separation line with maximum margin. Then, the professor adds one more point, which poses a challenge to linear separability. The solution proposed is to find a line that linearly separates the points while minimizing the number of misclassified points. However, this solution may not work in all cases, such as when there is a ring of points around the other points. To address this, the professor introduces a transformation function that expands the dimensions of the points. This function allows for the creation of new dimensions based on the existing ones. Although this expansion does not introduce new information, it facilitates the calculation of similarity between points and becomes useful in solving the quadratic program associated with SVMs.

1.4.11 What is the Output Question

The problem is to compute the dot product of two two-dimensional points, X and Y, by passing them through a function phi. The dot product X transpose Y is transformed into phi X transpose phi Y. The goal is to determine the output in terms of the components X1, X2, Y1, and Y2.

1.4.12 What is the Output Solution

In this lecture, the professor discusses the concept of output solution in the context of kernel methods and support vector machines (SVMs). The professor starts by explaining the transformation of input vectors (x) and output vectors (y) using a feature map function phi. The transformed vectors are then used to compute the dot product, which represents similarity between the vectors. The professor introduces the notion of similarity between vectors as a geometric concept, relating it to the equation of a circle. By transforming the data points into a higher-dimensional space, the professor demonstrates how it is possible to linearly separate the data points using a hyperplane. This technique, known as the kernel trick, allows for the use of a function (kernel) that represents similarity without explicitly performing the transformation. The squared dot product is used as the kernel function in the lecture. The professor also mentions that there are various other functions that can be used as kernels, and for each function, there is a corresponding transformation into a higher-dimensional space. This implies that the choice of kernel function is not limited, and almost any function can be used in practice. Overall, the lecture highlights the ability of kernel methods and SVMs to abstract the concept of similarity and transform data into higher-dimensional spaces, leading to improved classification and separation of data points.

1.4.13 Kernel

The lecture discusses kernel methods and support vector machines (SVMs) in machine learning. The speaker introduces the concept of using a kernel function to represent similarity between data points. The kernel function allows for injecting domain knowledge into the SVM algorithm without the need to compute points in a higher dimensional space. Different types of kernels, such as polynomial and radial basis kernels, are discussed and their properties explained. The lecture emphasizes the importance of capturing domain knowledge and presents examples of using kernel functions for discrete variables like strings, graphs, and images. The speaker also mentions the Mercer Condition, a technical requirement for kernel functions to ensure they behave like a well-behaved distance function.

1.4.14 Summary

Summary: Support Vector Machines (SVMs) were discussed in this lecture. The concept of margins in SVMs and their relation to generalization and overfitting was explained. The goal is to find a linear separator with the largest margin. The optimization problem for finding maximum margins was formulated as a quadratic program, and the dual of the quadratic program identified the support vectors. Support vectors are the necessary points from the input data for defining the maximum margin separator. SVMs were also connected to instance-based learning and ensemble methods. The kernel trick was introduced as a way to project data into a higher dimension and make comparisons using a similarity metric. Kernels need to satisfy the mercer condition, but in practice, some kernels that don't meet this condition still work. The lecture also mentioned the upcoming explanation of boosting and its relation to overfitting, which will be tied to the concepts learned about SVMs.

1.4.15 Back to Boosting

Boosting is a machine learning technique that does not always overfit in the same way as other algorithms. Boosting takes into account both error and confidence in its predictions. The final output of the boosted classifier is a weighted average of weak hypotheses, and the output is normalized between -1 and +1. Boosting increases confidence in correct predictions and decreases confidence in incorrect predictions, resulting in a larger margin between positive and negative examples. This larger margin helps to minimize overfitting. While boosting can be effective in avoiding overfitting, it is not guaranteed to never overfit.

1.4.16 Boosting Tends to Overfit Question

Boosting tends to overfit in several circumstances. The first possibility is when the weak learner always chooses the output that is closest to chance, even though it performs better than chance. The second possibility is when the weak learner used in boosting is a neural network with many layers and nodes. Another scenario is when there is a large amount of training data available for boosting. The fourth case is when the true underlying hypothesis is non-linear, making it more challenging to find a simple solution. Lastly, if boosting is allowed to train for an excessively long time, it is more likely to overfit.

1.4.17 Boosting Tends to Overfit Solution

In this lecture on kernel methods and support vector machines (SVMs), the speaker discusses the tendency of boosting to overfit solutions. The speaker begins by ruling out that boosting overfits only when it trains for too long, as demonstrated by a counterexample. They also dismiss the notion that boosting overfits specifically in nonlinear problems, as the linearity of the problem does not seem relevant to overfitting. The speaker explains that having a large amount of data actually mitigates overfitting, since there is more information to learn from. They also mention that if there is enough data to achieve zero training error, then there will also be zero generalization error. The speaker then discusses the possibility that boosting overfits because it uses a weak learner, specifically an artificial neural network with many layers and nodes. However, they are not entirely convinced of this explanation and mark it as a question for further consideration. The speaker moves on to the idea that boosting overfits when the weak learner chooses the weakest output. They clarify that boosting works as long as the weak learner outperforms chance (better than 50% accuracy), regardless of choosing the weakest or the strongest output. They provide an example of when boosting can overfit if the underlying learners (neural networks) overfit and get stuck in a loop of returning the same network. In this case, boosting is unable to address the overfitting issue. The speaker then engages

in a semantic discussion about the terms ”weak learner” and ”strong learner,” highlighting the difficulty in defining the latter. They suggest not getting too fixated on what a strong learner means when writing a proof. The lecture concludes with the mention of other cases where boosting tends to overfit, specifically in the presence of pink noise, which refers to uniform noise rather than Gaussian noise as in white noise. The speaker recommends further reading on pink noise and its relation to overfitting in boosting. The main takeaway is that if the underlying weak learner already overfits, it becomes challenging for boosting to overcome that overfitting.

1.5 Comp Learning Theory

1.4.18 Summary For Real

In this lecture, the instructor discusses support vector machines (SVMs) and how they connect to boosting. The focus is on understanding the theoretical construct of SVMs, specifically the concept of margins. The instructor suggests that they have covered all the necessary information about SVMs for now.

1.5.1 Learning Theory

This text discusses computational learning theory and its importance in addressing key questions about learning problems. The main goal is to define learning algorithms and determine if they work or not based on the problem definition. The lecture also explores the concept of upper-bound (improving algorithms) and lower-bound (fundamentally hard problems) analysis. Mathematical reasoning and careful definitions are necessary to answer these questions. Practical algorithms are discussed to illuminate the fundamental learning questions and the efficacy of certain algorithms. The lecture draws a parallel between computational learning theory and analyzing algorithms in computing.

1.5.2 Resources in Machine Learning Question

In computational learning theory, algorithms are analyzed in terms of their use of resources, typically time and space. An algorithm's running time is expressed in terms of n log n, while the amount of memory it takes up is measured as n-squared space. In the context of computational learning theory, it is important to analyze the resources used by algorithms in order to select the most efficient ones.

1.5.3 Resources in Machine Learning Solution

In this lecture, the speaker discusses the resources that need to be managed in a learning algorithm. The two main resources mentioned are time and space. It is important to analyze algorithms in terms of their efficiency in these two aspects. The speaker emphasizes the importance of choosing algorithms that run in shorter amounts of time, especially when faced with exponential time complexities. Similarly, algorithms that require a large amount of space may not be useful. In addition to time and space, the speaker introduces data as the third important resource in machine learning. The availability and quality of training samples greatly impact the effectiveness of a learning algorithm. The speaker acknowledges that the term ”samples” is interchangeable with ”data” or ”examples.” The goal is to learn well with a small amount of samples, as this indicates effective generalization and superior learning capabilities.

1.5.4 Defining Inductive Learning

Inductive learning is defined as learning from examples. When defining an inductive learning problem and evaluating an inductive learning algorithm, there are several important quantities to consider. These include the probability of success, the number of examples used for training, and the complexity of the hypothesis class. The complexity of the hypothesis class can impact the ability to learn complex concepts and may also increase the risk of overfitting. The accuracy to which the target concept is approximated, known as epsilon, is another important factor in understanding the complexity of a learning algorithm. Additionally, the way training examples are presented and selected for presentation can affect the learning process. Examples can be presented in a batch format, where a fixed training set is provided, or in an online format, where examples are presented one at a time and the algorithm predicts the label before receiving the correct answer. These different presentation and selection methods can result in different algorithm behaviors.

1.5.5 Selecting Training Examples

The lecture discusses the importance of selecting training examples in machine learning. Different methods of selecting training examples are explored, including the learner asking questions of the teacher and the teacher providing x, c(x) pairs. The concept of training examples coming from an underlying distribution is also discussed. The lecture emphasizes that these different methods of selecting training examples are crucial in understanding machine learning. The lecture concludes by referring back to the concept of 20 questions.

1.5.6 Teaching Via 20 Questions Question

In this lecture, the speaker discusses the concept of teaching via 20 questions in the context of machine learning. The learner's goal is to find the best hypothesis in a given class H, where each hypothesis maps inputs to either yes or no. The analogy used is that the hypothesis class H represents a set of possible people in a 20 questions problem, and X represents the set of questions that can be asked. The learner obtains information about the right answer through these questions, but no single question is revealing enough to determine the hypothesis. The lecture explores two scenarios: one where the teacher selects the questions for the learner to ask, and the other where the learner comes up with the questions. It initially seems like there shouldn't be a difference since the learner will ask the questions and the teacher will answer truthfully in both cases. However, the key distinction is that the teacher knows the answer, whereas the learner does not. A quiz is proposed to determine how many questions are necessary for a smart learner to identify the right person, assuming that the teacher provides good questions for the learner.

1.5.7 Teaching Via 20 Questions Solution

In this lecture, the concept of teaching via 20 questions is discussed. The teacher, who knows the correct answer, wants the learner to guess it as quickly as possible. The strategy discussed is for the teacher to choose a question that eliminates as many incorrect hypotheses as possible. This strategy provides the most information. An example is given where the teacher asks a series of questions to guess that the person in question is Michael Jordan, a professor at Berkley and a machine learning expert. It is noted that with a helpful teacher, the correct answer can be obtained in one question. However, if the learner is asking the questions, it becomes a more difficult problem. Asking if a specific name is the correct hypothesis is likely to be unhelpful. It is mentioned that a truly helpful teacher could potentially change the target, but this would be considered cheating.

1.5.8 The Learner Question

The learner faces a more difficult situation compared to the teacher when choosing which questions to ask because it does not know the right answer. The learner cannot determine in advance whether a question will have a yes or no answer, making it challenging to whittle down the set of possible answers. The number of questions the learner needs to ask depends on the size of the set of people to choose from. There are four formulae discussed: the size of the set, the logarithm of the size, two to the power of the size, and the possibility of obtaining the right answer in a single question. The lecturer asks Charles if he understands and can work through the problem. Charles confirms that he can.

1.5.9 The Learner Solution

In this lecture, the instructor discusses the concept of the learner solution in machine learning. The goal of the learner is to ask questions that provide the maximum amount of information to eliminate as many hypotheses as possible. The instructor emphasizes that the learner does not know the answer to the question, so they should focus on questions that can whittle down the number of possible answers. The expected number of hypotheses that can be eliminated with a question is about half of them, assuming the target hypothesis was chosen uniformly at random. The instructor also discusses the probability of going down each branch of a question and emphasizes the importance of choosing questions that split the set in half. By continually asking such questions, the learner can reduce the number of possible answers to one, which takes a logarithmic amount of time. The instructor concludes by noting that while this is larger than one, it is still a significant improvement in narrowing down the hypotheses.

1.5.10 Teacher With Constrained Queries

The lecture discusses the concept of a teacher with constrained queries in computational learning theory. The teacher typically asks questions to lead the learner to the correct hypothesis, but in this case, the teacher is limited in the types of questions it can ask. The lecture provides examples using a hypothesis class consisting of literals and their negation, where k-bit inputs are evaluated. The lecturer demonstrates how the teacher can use specific inputs to determine the output of the hypothesis. Not all input variables are relevant, and the teacher must consider the constraints when assessing the validity of the hypothesis.

1.5.11 Reconstructing Hypothesis Question

In this lecture, the instructor presents a table of examples with input patterns and actual output patterns of a hypothesis. The goal is for the students to figure out the hypothesis without being explicitly told. To do this, they need to determine whether each variable appears in the conjunction in its positive form, negative form, or not at all. The instructor mentions that the examples were specifically chosen to give Charles Isbell, a PhD student in the class, the best chance of finding the hypothesis with the fewest number of examples. The students are then instructed to start the task.

1.5.12 Reconstructing Hypothesis Solution

The lecture discusses the process of reconstructing a hypothesis solution in machine learning. The lecturer starts by examining the first two examples and identifying inconsistent variables. Variables with different values in the two examples are deemed irrelevant. Variables with the same values in both examples are considered necessary. The lecturer eliminates irrelevant variables (x1 and x3) from consideration. They then focus on determining the necessity of variables x2, x4, and x5. Using the first two examples, the lecturer initially thinks the answer is "not x2, x4, not x5" as it is consistent with those examples. They then validate this hypothesis with the next three examples. The lecturer demonstrates that flipping any of these variables from their original values results in an incorrect answer, confirming their necessity. The lecturer summarizes the two steps involved: identifying irrelevant variables and validating the necessity of remaining variables. The number of queries required for each step depends on the number of variables used in the formula (in this case, three variables). The lecturer concludes by noting that the total number of hypotheses is three to the power of k, where k is the number of variables. However, a smart teacher can reconstruct the solution with only k+2 queries, showing the efficiency of this approach. The lecturer poses the question of how many queries would be needed if the teacher was not available.

1.5.13 Learner With Constrained Queries

In this lecture on learner with constrained queries, the speaker explores the challenges faced by the learner in trying to learn without knowing the actual answer. It is discussed that there are 3 to the power of k possible hypotheses, and using the 20 questions trick, the learner could eliminate half of them in a linear time complexity. However, finding a specific question that can eliminate a third of the hypotheses is difficult and time-consuming. The speaker provides an example where the learner is asked to find a specific pattern and highlights the exponential amount of time it takes. It is mentioned that the cheat the learner has is knowing all the hypotheses, but this does not help in this case of constrained queries. The speaker acknowledges the frustration of not being able to ask the desired questions that could split the hypothesis class in half, and explains that most questions give little information, resulting in a time complexity of 2 to the power of k before finding the hypothesis. The speaker also mentions the optimality of the 20 questions approach when the learner knows nothing, but acknowledges that sometimes it can do worse or better. The constrained set of questions is identified as the main challenge, and it is stated that there is no way to approximate it. The speaker explains that queries need to be data points and that asking about specific properties of the hypothesis is not possible within the constraints. It is noted that getting a positive result is crucial to gaining knowledge, but the presence of negation makes most queries provide useless information. Despite the challenges, the speaker suggests that there may be other questions with happier answers, without elaborating further.

1.5.14 Learner With Mistake Bounds

In this lecture on machine learning, the instructor discusses the concept of learner with mistake bounds. In this learning formalism, the learner receives input and guesses the answer for that input. If the guess is

wrong, the learner is charged a point and informed of the mistake. The goal is to bound the total number of mistakes made by the learner. The algorithm presented starts off with a formula that assumes every variable is present in both positive and negated forms. The learner continues to guess "false" until it receives an input where the correct answer is "true" and makes a mistake. Based on this mistake, the learner eliminates certain variables from the formula. The process continues until the learner can predict the correct answer based on a specific bit pattern. The instructor explains that even if the learner sees an exponential number of examples, it will never make more than k + 1 mistakes. Consequently, a good teacher can provide examples that allow the learner to eliminate variables and ultimately learn the correct formula. The instructor notes that if the teacher knows how the learner starts out with the formula, they can give exactly the right number of examples to facilitate learning.

1.5.15 Definitions

The lecture discusses three different ways of choosing inputs for a learner: the learner chooses examples, the teacher chooses examples, and examples are given by nature. The lecture focuses on the fourth case, where nature chooses examples, which is the most interesting and complex. The lecture introduces important terms such as computational complexity, sample complexity, and mistake bound. Computational complexity refers to the computational effort necessary for a learner to converge to the right answer. Sample complexity refers to the size of the training set needed for the learner to create a successful hypothesis. In the case of nature choosing, sample complexity is the most relevant concept. The lecture also touches on the concept of finding the best hypothesis in a limited hypothesis class and distinguishes between computational complexity and sample complexity, with a focus on the latter.

1.5.16 Version Spaces

The concept of a version space is important for understanding algorithm analysis. The version space is the set of hypotheses that are consistent with a given training set. A learner is considered consistent if it produces a hypothesis that matches the true concept. The version space consists of all hypotheses that are consistent with the data.

1.5.17 Terminology Question

The lecture discusses the concept of a version space by providing an example. The target concept, C, is a mapping from two input bits to an output bit, specifically the XOR function. However, the training data only includes a few examples. The lecture then introduces a hypothesis set, which includes various functions such as copying or negating the inputs, ignoring the inputs and returning true or false, taking the OR, AND, or XOR of the inputs, and checking if the inputs are equal. The task is to determine which functions from the hypothesis set are in the version space for the given training set.

1.5.18 Terminology Solution

In this lecture, the speaker and Charles discuss the concept of being in the version space, which means being consistent with the given data. They go through different variables, such as X1 and X2, and determine if they are consistent with the data. They also discuss different logical operations, such as OR, AND, and XOR, and analyze their consistency with the given data. They conclude that X1, OR, AND, and XOR are consistent with the data, while the opposite of X1 and the opposite of X2 are not consistent. They also note that true and false are both inconsistent with the data, but not equivalent is consistent with the data as it represents XOR.

1.5.19 Error of h

The lecture discusses the concept of PAC (Probably Approximately Correct) learning. The two types of errors in learning are the training error and the true error. The training error measures the fraction of examples classified incorrectly by a hypothesis H on the training set. The true error measures the fraction of examples misclassified on a sample drawn from a distribution D. The true error is the probability of a sample being misclassified by a hypothesis. The true error takes into account misclassifications of examples that will never be seen and penalizes them proportionally to their probability.

1.5.20 PAC Learning

PAC Learning is about learning a concept from a concept class C. The learner L considers a hypothesis space H with N different hypotheses. The distribution of inputs is represented by D. Epsilon represents the desired error goal, where the hypothesis produced should have an error no bigger than epsilon. Delta represents the uncertainty goal, where the algorithm needs to work with a probability of one minus delta. Delta allows for some uncertainty because training examples are sampled from D, and it's possible to draw samples that only give one example repeatedly, leading to high error. PAC stands for "probably approximately correct," as we aim to be approximately correct, but we can't be exactly correct all the time, so we probabilistically aim for approximation.

1.5.21 PAC Learning Two

PAC-learnable refers to the ability of a learning algorithm to output a hypothesis with high accuracy and confidence. In order for a concept class to be PAC-learnable, the algorithm must output a hypothesis with an error rate less than or equal to a specified value. Additionally, the algorithm should be able to achieve this level of accuracy with a relatively small amount of time and a limited number of samples from the distribution. PAC-learnability allows for some flexibility in error rates and confidence levels, but it does not guarantee perfect error-free results.

1.5.22 PAC Learnable Question

The lecture discusses the concept class and hypothesis class, which are the same in this case. The concept class is the set of functions that return the Ith bit of an input, and the hypothesis class is the set of hypothesis functions that return the corresponding bit for each input. The goal is to determine if there exists an algorithm L that can PAC learn the concept class using the hypothesis class.

1.5.23 PAC Learnable Solution

The lecture discusses the concept of PAC learnability in machine learning. The goal is to determine if a learning algorithm can return the right answer or a close enough answer with limited data. The lecturer proposes a learning algorithm called the version space, which keeps track of all hypotheses consistent with the data seen and selects one uniformly when sampling stops. However, the lecture acknowledges the need to establish a bound on the number of samples required for a good answer and argues that more concepts and analysis are needed to determine the feasibility of the algorithm. The lecture concludes that the answer is likely yes, but further exploration is required.

1.5.24 Epsilon Exhausted

Epsilon exhaustion is a key concept in machine learning theory. It refers to a version space derived from a sample that is considered epsilon exhausted if all hypotheses in that version space have low error. In other words, if all the hypotheses left in the version space have error less than epsilon, the algorithm will work fine. If this condition is not met, the algorithm will be in trouble as there is a chance of selecting a hypothesis with high error. To ensure that only hypotheses with low error remain in the version space, some theoretical development is needed. The idea is that a version space is epsilon exhausted if everything that could be chosen from it has an error less than epsilon. If there is any hypothesis with error greater than epsilon, the version space is not epsilon exhausted.

1.5.25 Epsilon Exhausted Quiz Question

In this lecture, the concept of epsilon exhaustion is explained and applied in a quiz. The example involves a target concept and training data with specific outputs. A distribution D is also introduced, which assigns probabilities to different input examples. The goal is to find an epsilon value that exhausts the version space of the training set. The students are asked to work through this quiz question.

1.5.26 Epsilon Exhausted Quiz Solution

The lecture discusses the concept of epsilon exhausted version space in machine learning. It explains that setting epsilon to one is valid because it indicates that there is no hypothesis in the version space with an error greater than one. The lecture also discusses the computation of error for different hypotheses using training examples. It demonstrates the calculation of error for two hypotheses, x1 and or. It shows that x1 has an error of 0.5, while or has an error of 0. The lecture concludes by stating that epsilon is the smallest value that ensures epsilon exhaustion and if epsilon were smaller than 0.5, the version space would not be epsilon exhausted.

1.5.27 Haussler Theorem

Haussler's Theorem is a way of bounding the true error of hypotheses in a hypothesis set based on the number of training examples. The goal is to determine how much data is needed to identify hypotheses with high true error. The theorem calculates the probability that a hypothesis matches the true concept using a given dataset, which is less than or equal to one minus epsilon, where epsilon represents the error. The probability that a hypothesis remains consistent with the true concept after drawing m examples is calculated as one minus epsilon raised to the power of m. If there are k hypotheses with high true error, the probability that at least one of them remains consistent is calculated as one minus epsilon to the power of m times k. The upper bound on the number of bad hypotheses is k, which is also bounded by the total number of hypotheses in the set. The expression is then further simplified to a more convenient form.

1.5.28 Haussler Theorem Two

The lecturer explains an important step in the derivation and states the relationship between negative epsilon and the natural log of one minus epsilon. They explain that minus epsilon is a straight line with a slope of minus one, while the log of one minus epsilon falls below this line. This relationship is used to derive an upper bound on the hypothesis space, which is more convenient to work with. The lecturer then discusses how this upper bound relates to the failure probability delta and concludes that the sample size M needs to be at least as large as one over epsilon times the logarithm of the hypothesis space size plus the logarithm of one over delta. This result shows that the sample size requirements are polynomial in epsilon, delta, and the hypothesis space size. The lecturer emphasizes the practical significance of this result, stating that as long as the size of the hypothesis space and the desired targets for epsilon and delta are known, sampling a sufficient amount will yield satisfactory results.

1.5.29 PAC Learnable Example Question

The lecture discusses a question related to PAC learning in machine learning. The hypothesis space consists of functions that take 10-bit inputs, with each hypothesis corresponding to returning one of the 10 bits. The goal is to find a hypothesis with an error less than or equal to 0.1 and a failure probability less than or equal to 0.2. The input distribution is assumed to be uniform. The lecture aims to determine the number of samples needed to PAC learn this hypothesis set using a specific algorithm.

1.5.30 PAC Learnable Example Solution

The lecture discusses the sample complexity of learning a hypothesis space. It introduces the equation $M \geq 1/\epsilon * ln(H) + ln(1/\delta)$, where M is the number of samples needed, epsilon is the desired error rate, H is the size of the hypothesis space, and delta is the confidence level. The lecture demonstrates how to calculate the number of samples needed for a specific problem, using an example where $H = 2^10$ and the desired error rate is 10%. It concludes that 40 samples are needed for this problem. The lecturer also mentions that the sample complexity is not affected by the distribution of the samples, due to cancelation of the distribution between the training and true error. It suggests that if a lower error rate is desired, more samples will be needed.

1.6 VC Dimensions

1.5.31 What Have We Learned

The lecture discussed computational learning theory and its application in understanding the learnability of concepts in machine learning. The instructor and student discussed the role of teachers and learners in the learning process, focusing on the concept of sample complexity and the amount of data required for learning. The learner's ability to ask questions and the teacher's role in selecting questions were also explored. Different scenarios, including when the teacher is nature, were considered. The lecture also introduced the concept of mistake bounds and discussed version spaces and PAC (Probably Approximately Correct) learnability. The distinction between training error, test error, and true error was explained, with true error being connected to the distribution of the data. An equation for sample complexity bound was presented, which polynomially depends on the size of the hypothesis space, target error bound, and failure probability. The lecture briefly mentioned the agnostic learning scenario, where the learner does not need to find an exact hypothesis that matches the target concept but one that approximates it well. The lecture also touched upon the challenges of infinite hypothesis spaces and highlighted the need for further discussion on this topic in future lessons.

1.6.1 Infinite Hypothesis Spaces

The lecture discusses bounding the number of samples needed to learn a classifier or concept in a given hypothesis base. The formula for this is found to be $samples \geq 1/\epsilon * log(hypotheses) \div log(1/\delta)$, where ϵ represents the error parameter, and δ represents the failure parameter. If we want a very low failure probability (small δ) or a small error (small ϵ), we will need more samples. The concern raised is about the size of the hypothesis space and what happens if it is infinitely large. The lecturer suggests taking a quiz to explore this further.

1.6.2 Which Hypothesis Spaces Are Infinite Question

The text is a transcript of a lecture on the topic of hypothesis spaces and their importance in machine learning. The lecturer starts by stating that they will discuss the significance of considering infinite hypothesis spaces. They then proceed to list several hypothesis spaces from previous lectures and instruct the students to determine whether each hypothesis space is infinite or not.

1.6.3 Which Hypothesis Spaces Are Infinite Solution

In this lecture, the speaker discusses the nature of hypothesis spaces and whether they are finite or infinite. They start by considering linear separators, artificial neural networks, decision trees with discrete inputs, and decision trees with continuous inputs. They determine that all of these hypothesis spaces are infinite. Then, they discuss k-nearest neighbors (k-nn) and debate whether its hypothesis space is finite or infinite. One argument suggests that there are an infinite number of k-nn classifiers based on the layout of the data points, while the other argument states that there is only one classifier depending on the fixed training set. The speaker concludes that the hypothesis space of k-nn is subject to interpretation and may be considered either finite or infinite. They note that k-nn is sometimes referred to as non-parametric, despite technically having an infinite number of parameters. Lastly, they briefly mention that other methods, such as neural networks and decision trees, also result in multiple classifiers consistent with the given data.

1.6.4 Maybe It Is Not So Bad

In this lecture on VC dimensions, the instructor presents an example to demonstrate that the situation may not be so bad after all. The example involves an input space of the first 10 integers and a hypothesis space consisting of whether an input is greater than or equal to a certain threshold (theta). The instructor emphasizes that the hypothesis space is infinite due to theta being a real number. However, the instructor proposes the idea of using the algorithm discussed previously to learn from this space. The algorithm involves keeping track of all hypotheses and the version space, and using randomly drawn examples to exhaust the version space and select a suitable hypothesis. The challenge arises from the fact that there are infinitely many hypotheses in the space. However, the instructor suggests that if theta were a positive or non-negative integer, it would simplify the situation, even though there would still be an infinite number

of hypotheses. The reason is that the input space is finite, so any value of theta greater than 10, for example, would yield the same answer and could be ignored. Therefore, by only considering non-negative integers 10 or below as hypotheses, the instructor argues that the hypothesis space becomes finite and provides the same results as tracking the infinite space. The distinction between the syntactic and semantic hypothesis space is explained, where syntactically infinitely many functions can be represented by a finite set of semantically different functions. The example of a decision tree with discrete inputs is also mentioned to illustrate this concept. This example lays the foundation for discussing learning in more complicated hypothesis spaces without the need to track an infinite number of hypotheses. By recognizing that not all hypotheses are meaningfully different, it becomes possible to explore and understand more complex spaces without getting overwhelmed by indefinitely expanding possibilities.

1.6.5 Power of a Hypothesis Space

The power of a hypothesis space can be measured by considering the largest set of inputs that the hypothesis class can label in all possible ways. In an example with only one input, there are two possible ways to label it, true or false. However, there is no pair of inputs that can be labeled in all four ways, making the hypothesis space weak. This measure of power applies universally and provides insight into the expressiveness of the hypothesis space.

1.6.6 What Does VC Stand For

The concept of VC dimensions is applicable in various contexts with infinite hypothesis classes. It is often referred to as "shattering". The VC dimension is the size of the largest set of inputs that the hypothesis space can shatter. The VC stands for Vapnik-Chervonenkis, who established the definition and its connection to the amount of data required for effective learning. This dimensionality enables us to determine the amount of data needed for learning, even when the hypothesis class is infinite. It would be beneficial to examine different types of hypothesis classes and measure their VC dimensions.

1.6.7 Internal Training Question

The lecture discusses the concept of VC Dimensions in machine learning, using a concrete example of a hypothesis space consisting of intervals on the real line. The goal is to determine the VC dimension of this hypothesis space, which represents the largest set of inputs that can be labeled in all possible ways using hypotheses from the given space. The lecture prompts the students to calculate the VC dimension and write it as an integer in a designated box.

1.6.8 Internal Training Solution

The lecture discusses the concept of VC dimensions in machine learning. The instructor explains the process of determining whether the VC dimension is at least one, using the example of a number line. They demonstrate that it is possible to label a single point as positive or negative by placing brackets around it. They then extend this process to determine if the VC dimension is at least two by placing two points on the line and considering the various labelings. The instructor shows that all four possibilities (plus plus, minus plus, plus minus, and minus minus) can be achieved by appropriate bracket placements. However, the instructor explains that when considering three points on the line, there is no way to arrange them such that all possible labelings can be achieved. Consequently, the VC dimension in this case is two. The instructor emphasizes the importance of providing an example that can shatter the points to prove a lower bound and explains the difference between proving lower and upper bounds using existential and universal quantifiers. The instructor also mentions the application of DeMorgan's Law in logical reasoning.

1.6.9 Linear Separators Question

Linear separators are important in machine learning, unlike intervals. The VC dimension for linear separators can be easily determined in two-dimensional space. A linear separator is defined by a weight parameter, w, which is multiplied by the input and compared to a threshold, theta. If the result is greater than or equal to theta, it is a positive example; otherwise, it is negative. This creates a line that separates positive and negative examples. The lecture does not explain the VC dimension and suggests that further information will be provided.

1.6.10 Linear Separators Solution

The lecture begins with a discussion on the VC dimension of linear separators. The instructor systematically examines whether the VC dimension is greater than or equal to 1, 2, 3, and 4 by giving examples. It is established that the VC dimension is equal to 1 because a point on the x-axis can be labeled positive or negative by flipping weights. Similarly, for a VC dimension of 2, two points on a line can be labeled in four different combinations. Next, the instructor introduces three points on a number line and demonstrates that a linear separator cannot separate them. To overcome this challenge, the instructor suggests moving one point off the number line to create a triangle, which allows for separation. It is shown that all labeling combinations can be achieved using a vertical line. The lecture concludes with a discussion on the VC dimension of 4. Using a diamond shape formed by four points, it is argued that a linear separator cannot separate them due to crossing lines, mimicking the behavior of an XOR function. The instructor mentions that regardless of the arrangement of points, there is always a labeling that cannot be achieved in the hypothesis class. As a result, it is determined that the VC dimension of linear separators is 3. The instructor poses a question regarding the VC dimension in higher-dimensional spaces, but does not explore it further.

1.6.11 The Ring

The lecture discusses the concept of VC dimensions in machine learning. In particular, the focus is on the relationship between the dimensions of different hypothesis spaces and the number of parameters needed to represent them. The lecturer provides examples in one, two, and three dimensions, illustrating that the VC dimension is equal to the number of parameters required plus one. For a d-dimensional hyperplane concept or hypothesis class, the VC dimension is d + 1. This is because d parameters are needed for the weights in each dimension, plus one for the threshold value.

1.6.12 Polygons Question

The lecture discusses a quiz on convex polygons, with the hypothesis that X is an R squared and the points inside the polygon are considered ”inside” for the purpose of the discussion. The speaker poses a question about the VC dimension of convex polygons and suggests allowing the students to answer before proceeding with the quiz.

1.6.13 Polygons Solution

The lecture explores the concept of VC dimensions in the context of convex polygons. The lecturer discusses how the number of parameters for convex polygons is infinite, due to the unbounded number of sides a polygon can have. The lecture suggests that the VC dimensions of convex polygons may also be unbounded, but notes that circles, which are a limit case of polygons, have a VC dimension of three. The lecture then introduces the idea of shattering points on a circle with convex polygons. By connecting the points in different ways, all possible labellings can be achieved. The lecture demonstrates that with six points on a circle, all labellings are achievable, and explains how to exclude specific points from being labeled positive by not connecting them in the polygon. It is concluded that the VC dimension of this hypothesis class is infinite, as the number of points that can be captured in this way is unbounded. The lecture clarifies that the polygons used in the example are convex, as they are confined within the unit circle. The lecture highlights the connection between the infinite VC dimension and the number of parameters in the hypothesis class. The lecture concludes by addressing the question of the natural log of an infinite hypothesis space and its relation to VC dimensions.

1.6.14 Sample Complexity

The lecture discusses the connection between VC dimension and sample complexity in machine learning. The equation for sample complexity is presented, which states that the size of the training data should be at least as big as a certain expression to achieve a desired error rate with confidence. The expression includes the VC dimension of the hypothesis class, which determines the amount of data needed. It is noted that the VC dimension plays a similar role to the natural log of the size of the hypothesis space in the finite case. The lecture also mentions the difference between the size of the hypothesis space and the dimension of it. The VC dimension of a finite hypothesis class is discussed as well.

1.6.15 VC of Finite H

The VC dimension of a finite hypothesis class, denoted as H, can be determined by considering an upper bound. If the VC dimension is D, then there must be at least 2^D distinct concepts in H. This is because each distinct labeling requires a unique hypothesis in the class. Therefore, 2^D is less than or equal to the number of hypotheses in H. By manipulating this inequality, we find that D is less than or equal to the logarithm base 2 of the size of H. This logarithmic relationship between the size of a finite hypothesis class and its VC dimension can also be observed in the other direction. It is important to note that if the VC dimension is finite, H is PAC-learnable. Conversely, if the VC dimension is infinite, H cannot be learned. The VC dimension captures the concept of PAC-learnability in a concise manner.

1.7 Bayesian Learning

1.6.16 Summary

In this lecture on "Bayesian Learning," the instructor and student discuss the key concepts of VC dimension. They learned that VC dimension captures the notion of shattering and the relationship between VC dimensions and parameters. They also discussed the impact of adding additional nodes to a neural network's hidden layer on the VC dimension. They noted that VC dimension is not only related to the number of parameters but also to the true number of parameters. The lecture also covered how VC dimension relates to the size of the hypothesis space for finite hypothesis spaces, as well as its relation to sample complexity. The instructor also mentioned the importance of computing the VC dimension by providing an example for the lower bound and proving an upper bound. Lastly, they discussed how VC dimension relates to PAC Learnability.

1.7.1 Intro

Bayesian Learning is a framework for learning in machine learning that aims to find the most probable hypothesis given data and domain knowledge. This is achieved by finding the hypothesis with the highest probability, which can be represented as the probability of a hypothesis h drawn from a hypothesis class given some data D. The goal is to find the argmax of h, which is the hypothesis with the highest probability given the data. The lecture will explore this expression in detail.

1.7.2 Bayes Rule

The lecture discusses Bayes' rule and its importance in understanding and finding the most probable hypothesis given data. Bayes' rule is derived from the chain rule in probability theory and relates the probability of a hypothesis given data to the probability of data given the hypothesis. This rule allows us to calculate the likelihood of seeing data given a specific hypothesis, which is easier to compute than the original probability of the hypothesis given the data. The lecture also explains the meaning of other terms in Bayes' rule, such as the probability of the data and the probability of the data given the hypothesis. The probability of the data represents our prior belief about the likelihood of seeing a particular set of data, and the probability of the data given the hypothesis measures the likelihood of observing certain labels given a set of inputs. The lecture uses an example to illustrate how the probability of the data given the hypothesis can be calculated by running the hypothesis and assigning probabilities to labels.

1.7.3 Bayes Rule p2

Bayesian Learning is a method that incorporates prior knowledge or beliefs about hypotheses in order to make predictions. The probability of a hypothesis is viewed as a prior belief about its likelihood compared to other hypotheses. Each prior belief represents domain knowledge, such as similarity metrics for KNN or beliefs about the structure of neural networks. The prior probability over hypotheses encapsulates all of this prior knowledge. Bayes' rule can be used to update the probability of a hypothesis given new data. The probability of a hypothesis can increase if the prior probability is higher, if the probability of the data given the hypothesis is higher (indicating higher accuracy), or if the probability of the data decreases. Bayes' rule provides a way to find the best hypothesis by explicitly defining it as the probability of the hypothesis given the data.

1.7.4 Bayes Rule Quiz Question

Summary: In this lecture, the instructor presents a quiz question related to Bayesian learning. The scenario involves a man who visits his doctor due to back pain. The doctor administers a lab test that is known to be quite accurate. The test has a 98% correct positive rate, meaning that if a person has the disease being tested for, the test will correctly identify it 98% of the time. The test also has a 97% correct negative rate, meaning that if a person does not have the disease, the test will correctly identify it 97% of the time. The disease being tested for is called "spleentitis," which is extremely rare. Only a fraction of the population has this disease. Despite its rarity, the lab test returns a positive result for the man. The quiz question is whether the man actually has spleentitis based on the positive test result. The instructor asks for clarification on the percentages, and clarifies that 0.008 is not a percentage. The lecture ends with the audience being given time to think about the answer to the quiz question.

1.7.5 Bayes Rule Quiz Solution

The lecture discusses the application of Bayes' Rule to analyze the probability of a patient having spleentitis based on a test result. It emphasizes the probabilistic nature of the test and the need to consider prior probabilities. The lecture explains that Bayes' Rule calculates the probability of the hypothesis given the data and involves the probability of the data given the hypothesis, the prior probability of the hypothesis, and the probability of the data. The lecture discusses the importance of priors and how they can influence the outcome. It also mentions that changing the population being tested can affect the priors and make the test more useful. The lecture concludes by raising questions about the threshold at which a positive result becomes more convincing and discussing the philosophical implications of priors in determining the usefulness of tests.

1.7.6 Bayesian Learning

The lecture discusses an algorithm for Bayesian learning. The algorithm involves calculating the probability of each candidate hypothesis given the data, and outputting the hypothesis with the maximum probability. The lecture highlights that the denominator in the probability calculation can be ignored since it doesn't affect the argmax computation. The lecture also discusses the concept of prior probabilities and how they can be dropped if necessary. The lecture mentions that the algorithm is computationally challenging when the hypothesis space is large or infinite. However, the lecture emphasizes the importance of the algorithm in providing a gold standard for learning and comparing results to real-life algorithms. Examples are given to support this point.

1.7.7 Bayesian Learning in Action

The lecture discusses Bayesian learning and its application in machine learning. The lecturer sets up a generic problem for machine learning, where a labeled training data set is given. The assumptions are made that the data is noise-free, the true concept is within the hypothesis space, and a uniform prior is used. The goal is to compute the probability of a hypothesis given the data. The lecture breaks down the computation into three terms: prior probability, probability of data given hypothesis, and probability of data. The prior probability is one over the size of the hypothesis space. The probability of data given hypothesis is 1 if the labels match the hypothesis for all training examples, and 0 otherwise. The probability of data is the size of the version space over the size of the hypothesis space. The final result shows that the probability of a hypothesis being correct is uniform among the hypotheses in the version space. This algorithm is valid under the assumptions of noise-free data, the concept being in the hypothesis space, and a uniform prior. It is concluded that this approach is always the right choice in Bayesian learning.

1.7.8 Noisy Data Question

The lecturer presents a scenario with noisy data to illustrate its impact on hypothesis in machine learning. The data follows a pattern where the label, d of i, is equal to a constant, k, multiplied by the input, x of i. The probability of obtaining each multiple of x of i is given by 1 divided by 2 to the power of k. The lecturer explains that this distribution is chosen because the sum of all probabilities from one to infinity is equal to one. The noisy aspect of the data means that the hypotheses will not always agree with the labels due to a stochastic process corrupting the output. The lecturer provides a specific dataset and a candidate

hypothesis (the identity function), and asks the student to calculate the probability of observing this dataset if the hypothesis is true. The noise is applied independently to each input-output pair.

1.7.9 Noisy Data Solution

The lecture discusses the solution to handling noisy data in Bayesian learning. The hypothesis is that when a data item is inputted, it is affected by a noise process that multiplies it by a certain factor. The probability of this multiplication is one over two to the power of the multiplier. The lecturer explains how to calculate the probabilities for each data item based on the multiplier. The probabilities are then multiplied together to obtain the probability of the entire data set. It is emphasized that if the data item is not divisible by the multiplier, the probability is zero. The lecturer concludes by stating that the next step will involve applying this concept to derive a solution for handling noisy data.

1.7.10 Return to Bayesian Learning

In this lecture, the instructor discusses the concept of Bayesian learning in the context of real-valued functions. The instructor explains that in Bayesian learning, the goal is

1.7.11 Best hypothesis Question

The lecture begins with a quiz to test the understanding of using Bayesian learning. The main insight is that the answer to finding the sum of squared errors in Bayesian learning is independent of the hypothesis class and only dependent on the key assumptions made. The quiz presents three functions and asks which is the best one under the assumption that the data was generated by a deterministic function with added Gaussian noise. The functions include a constant function, a linear function, and a mod function. A uniform prior is assumed over these three hypotheses. The quiz participant is asked to determine the best function.

1.7.12 Best hypothesis Solution

In this lecture, the speaker discusses the process of finding the best hypothesis solution using squared error. They extend a given table to include the output for three different functions, and then compute the squared error for each function on the data. They choose the function with the lowest squared error as the best hypothesis. The speaker mentions writing a program to perform this computation. The first function they analyze is a constant function, which gives an error of 19. They then try the function using $x/3$, which gives an error of 19.4444. Next, they substitute 9 into the function and obtain an error of 12. Despite its odd shape, this function fits the data the best. The speaker suggests there might be a better linear function and performs linear regression to find the optimal function. The resulting function has an intercept of 0.959 and a slope of 0.165. The squared error for this function is 15.8, which is better than the $x/3$ function but worse than the constant function. The best constant function is found to be the mean of the data, which is 2.17. The squared error for this constant function is 18.8. The speaker concludes that sometimes using modular arithmetic, or "mod," can be useful when dealing with unusual data.

1.7.13 Minimum Description Length

In this lecture on Minimum Description Length (MDL) in Bayesian Learning, the speaker begins by introducing the concept of the maximum a posteriori equation, which determines the best hypothesis by maximizing an expression. They then apply the natural logarithm to both sides of the equation, in order to convert products into sums and simplify the equation. They use a logarithm base 2 for convenience. Next, they convert the maximum in the equation into a minimum by multiplying everything by -1. The speaker points out that the logarithm and probability in the equation are reminiscent of information theory and entropy. They explain that the optimal code for an event with probability P is given by minus the logarithm base 2 of P. They then apply this fact to the equation and conclude that in order to find the maximum a posteriori hypothesis, the goal is to minimize the two terms that can be described as lengths. The speaker clarifies that the lengths refer to the length of the probability of the data given the hypothesis and the length of the hypothesis itself. They explain that the length of the hypothesis is the number of bits required to represent it. The goal then is to minimize this length by assigning shorter codes to more likely hypotheses and longer codes to less common ones.

1.7.14 Which Tree Question

The lecture begins with a question regarding the length of two decision trees.

1.7.15 Which Tree Solution

Smaller decision trees, with fewer nodes and less depth, are preferred over bigger decision trees. The length of a decision tree can be seen as a measure of how easy it is to represent the tree. A Bayesian argument for Occam's razor and pruning is made, stating that smaller trees are more likely. The length of the data given a hypothesis captures the notion of how well the hypothesis fits the data. Minimizing error and the size of the hypothesis is important in finding the simplest hypothesis that explains the data, following the principle of Occam's razor. This tradeoff between error and complexity is known as the minimum description length. Algorithms have been developed to find the tradeoff between error and size. However, there are challenges in comparing the units of size and types of error. In neural networks, the complexity is not in the number of parameters directly, but in the representation of parameter values. Bayesian learning provides insights into the decisions made and validates approaches such as minimizing squared error and following Occam's razor.

1.7.16 Bayesian Classification Question

The lecturer presents a quiz about Bayesian classification. Three hypotheses, h1, h2, and h3, assign labels to a specific input x. The probabilities of h1, h2, and h3 given the data are 0.4, 0.3, and 0.3 respectively. The lecturer asks for the best label for x and the options are plus or minus. The answer to this question is not explicitly mentioned.

1.7.17 Bayesian Classification Solution

In Bayesian learning, the goal is to find the best hypothesis and the best label. The most likely hypothesis is the maximum a posteriori (MAP) hypothesis, while the most likely label is determined by calculating the probability of each hypothesis and letting them vote. Finding the best hypothesis is a simple algorithm of computing the probability for each hypothesis and selecting the maximum. However, finding the best label requires a weighted vote for each hypothesis based on the probability given the data. The probability of the label given the hypothesis is either one or zero, and the largest probability is chosen. This can be derived using rules of probability and Bayesian's rule. The expression that maximizes the most likely value is derived from this. Overall, finding the best hypothesis is a means to an end, with the actual focus being on finding the best value or label.

1.8 Bayesian Inference

1.7.18 Summary

In this lecture on Bayesian Inference, the instructor summarized the key concepts learned. These include Bayes rule, which allows for the computation of the probability of data given a hypothesis, making it easier to compute compared to the probability of a hypothesis given the data. The importance of priors and their impact on the inference process was also discussed. The lecture introduced the concepts of MAP (Maximum a posteriori) and ML (Maximum likelihood) hypotheses, with the former being obtained when the prior is uniform. The connection between maximum a posteriori and least squares was explored, providing justification for their use. The instructor also mentioned the Minimum Description Length story and its relationship to Bayesian learning. The lecture ended by discussing classification and Bayes classifiers, highlighting that voting of hypotheses is the Bayes optimal classifier. Lastly, the instructor expressed curiosity about inferring probabilities from various quantities and observations, requesting further research on the topic.

1.8.1 Intro

The lecture introduces the concept of Bayesian Networks as a representation for manipulating probabilistic quantities over complex spaces. The speaker discusses the significance of representing and reasoning with probabilistic quantities and explains that Bayesian Networks align with the topic of Bayesian learning

discussed in the previous lecture. The speaker also mentions the use of a different color scheme for the lecture.

1.8.2 Joint Distribution

In this lecture on Bayesian Inference, the concept of joint distribution is introduced. The speaker uses the example of storm and lightning to explain the idea. They ask the audience to estimate the probabilities of different combinations of storm and lightning occurring at a random day and time, specifically at 2 PM in the summer in Atlanta. The probabilities given are: storm and lightning both occurring (25%), storm without lightning (40%), lightning without storm (5%), and no storm or lightning (30%). These probabilities add up to 100%. The speaker emphasizes that this is an example of a joint distribution and suggests using a quiz format to further explore this concept.

1.8.3 Joint Distribution Quiz Question

In this lecture, the instructor presents a quiz on Bayesian inference. The quiz involves using given probabilities to answer questions related to a joint distribution. The first question asks for the probability of seeing no storm when looking out the window at 2 PM in the summer in Atlanta. The second question involves calculating the probability of lightning given that there is a storm. The instructor mentions that knowledge of conditional probability is necessary to solve the quiz.

1.8.4 Joint Distribution Quiz Solution

In this lecture, the speaker discusses how to calculate probabilities in a joint distribution. They use a specific example to demonstrate the process. The first question is about the probability of there not being a storm. The speaker explains that by looking at the cases where the storm is false, there are two of them, and by adding the probabilities of those cases, they obtain a probability of 0.35. The second question asks about the probability of lightning, given that there is a storm. The speaker uses a similar approach, looking at the cases where the storm is true. They point out that out of these two cases, only one includes lightning, and thus the probability of there being lightning given a storm is 0.25. However, to calculate the correct answer, they divide 0.25 by the probability of there being a storm, which is 0.65. This gives them a probability of 0.4615, or 5/13. The speaker concludes that it is less likely for there to be lightning when there is a storm, as it happens less than half the time. They justify this by stating that otherwise lightning would occur every time there is a storm, which is not the case. They also mention that there are often breaks between lightning occurrences.

1.8.5 Adding Attributes

When adding more variables to a scenario, the number of probabilities that need to be considered increases exponentially. For example, if there are 100 variables, the number of probabilities would be 2^100, which is an unimaginably large number. This becomes especially inconvenient when variables have multiple values. To address this, we can factorize the joint distribution into smaller pieces that can be recombined into larger pieces. This approach allows for a more convenient representation and computation of probabilities.

1.8.6 Conditional Independence

Conditional independence is the idea that a variable X is independent of another variable Y, given a third variable Z, if the probability distribution governing X is independent of the value of Y given the value of Z. This means that if the value of Z is known, the probability of X can be determined without considering Y. If this property holds for all possible combinations of values for X, Y, and Z, then X is considered conditionally independent of Y given Z. Conditional independence is similar to normal independence, where the joint distribution of two variables is equal to the product of their marginals. The difference is that conditional independence allows for the possibility that the probability of X given Y can also be represented by the probability of X, as long as there is a value of Z that satisfies this property. Conditional independence allows for more powerful factorization of probability distributions, as it allows for independence even in more general circumstances.

1.8.7 Conditional Quiz Question

The lecture introduces a quiz on conditional independence. The task is to find a combination of truth values for thunder and lightning (i.e., true/true, true/false, false/true, or false/false) that satisfies a specific condition. This condition states that the probability of thunder being true, given the assigned value of lightning and the storm being true, should be equal to the probability of thunder being true, given the assigned value of lightning and the storm being false. The lecture emphasizes that the value assigned to the storm does not impact the outcome. The bottom two boxes will be automatically filled based on the values assigned to the top two boxes.

1.8.8 Conditional Quiz Solution

The lecture discusses finding the conditional probability of a variable given other variables using a table. The speaker demonstrates this by using an example involving the variables thunder, lightning, and storm. The probability of thunder being true given that lightning is false and storm is true is calculated by dividing the probability of thunder being true by the probability of lightning being false and storm being true. The same calculation is done for the case when lightning is false and storm is false. In both cases, the resulting probability is 0.1, indicating that thunder is conditionally independent of storm given lightning. This means that the probability of thunder given lightning and storm is equal to the probability of thunder given lightning. The lecture concludes by mentioning that this property of conditional independence can be represented in a concise form.

1.8.9 Belief Networks Question

The concept of a belief network, also known as a Bayes Net, Bayesian Network, or graphical model, involves representing the conditional independence relationships between variables in a joint distribution graphically. This is done using nodes for variables and edges for explicit dependencies. In this case, the lecture discusses how to fill in the prior probability of storm and the probabilities of lightning and thunder, given storm and lightning or not lightning. These probabilities can be found by marginalizing out relevant variables. By determining these probabilities, any desired probability in the joint distribution can be calculated by multiplying corresponding components together. The lecture provides a quiz to fill in the values in a table, with the numbers copied from previous slides to aid in the task.

1.8.10 Belief Networks Solution

The lecture discusses the calculation of probabilities in a belief network using the example of storms, lightning, and thunder. It shows how to determine the probability of lightning given a storm and the probability of thunder given lightning. The lecture emphasizes that the arrows in the belief network do not represent causality but rather conditional independence. The speaker explains that the belief network is not a simple network but a graph with parent and child relationships, and the number of probabilities to keep track of grows exponentially with the number of parents. Finally, the speaker clarifies that the arrows represent conditional dependencies and not causal relationships.

1.8.11 Sampling From The Joint Distribution Question

In Bayesian inference, sampling from a joint distribution is a useful task. In order to sample from a Bayesian network, an appropriate order needs to be chosen. The order should ensure that the variables are sampled in a way that takes into account their conditional dependence relationships. By sampling from variables that have no incoming arrows first and then sampling from variables that are conditioned on previously sampled variables, a probability distribution can be effectively sampled. This process is referred to as sampling trick.

1.8.12 Sampling From The Joint Distribution Solution

In this lecture, the concept of topological sort in Bayesian inference is discussed. Topological sort is a graph theoretic property that is used to order nodes in a way that ensures variables with incoming links that haven't been visited are placed after the ones that have been visited. This sort is similar to alphabetical or lexicographic order but is specifically for directed acyclic graphs (DAGs) used in Bayesian networks. In

these networks, variables depend on other variables, but cyclic dependencies are not allowed. If there were cycles in the graph, the probability distribution would not be meaningful. Therefore, Bayesian networks are typically constrained to acyclic graphs. The addition of cycles in the graph would imply the absence of conditional independencies, which is not the desired model.

1.8.13 Recovering the Joint Distribution

In this lecture, we learn about recovering the joint distribution from conditional probability tables in Bayesian networks. By multiplying the individual values from the conditional probabilities for each variable, we can compute the probability of any joint combination of variables. This is a more compact representation compared to the standard representation, where we assign probabilities to all possible combinations of variables. The number of values needed to specify the joint distribution depends on the number of variables and their relationships. If the variables are completely independent, only a few numbers are needed. However, the exponential growth of the joint distribution depends on the number of parents each variable has in the network. The fewer the number of parents, the more compact the distribution becomes.

1.8.14 Sampling

Sampling is useful in Bayesian inference because it allows for the generation of values consistent with a given distribution. This can be helpful in simulating complex processes and understanding patterns in data. Sampling can also be used for approximate inference, where instead of complex probability calculations, a set of possible worlds is imagined to determine the frequency of certain outcomes. Additionally, sampling can aid in visualization and provide a more intuitive understanding of data, especially in cases where there are numerous variables with complicated relationships. Approximate inference is used when exact calculations are difficult or time-consuming.

1.8.15 Inferencing Rules

In this lecture on Bayesian inference, the instructor introduces some rules of probability that are useful for conducting inferencing. The first rule is marginalization, which involves representing the probability of a value by summing over another variable and considering the joint probabilities. The instructor explains that the probability of variable X can be broken down into the probability of X when variable Y is false, plus the probability of X when variable Y is true. The second rule discussed is the chain rule, which states that the probability of variables X and Y can be written as the probability of X multiplied by the probability of Y given X. The instructor highlights the importance of including the "given X" part, as it indicates that the variables are not independent. However, if the "given X" part is dropped, it implies that the variables are completely independent and their product can be considered. The instructor also mentions that the order of variables on the left side of the equation doesn't matter. For example, the probability of X times the probability of Y given X is equivalent to the probability of Y times the probability of X given Y. The lecture then prompts a quiz, but the details and questions of the quiz are not provided.

1.8.16 Inferencing Rules Quiz Question

The lecture discusses the concept of Bayes Nets and the chain rule. The speaker explains that the probability of X and Y can be expressed as either the probability of X multiplied by the probability of Y given X, or as the probability of Y multiplied by the probability of X given Y. The lecture poses a question asking which of these expressions corresponds to a specific network.

1.8.17 Inferencing Rules Quiz Solution

In this lecture, the speaker discusses the concept of Bayesian inference. They start by clarifying that in the equation P(y) x P(x—y), the probability of y does not depend on any other variable, making the second graph the correct one. The speaker also explains that the second factor in the equation represents the probability of x given y, with an arrow from y to x indicating the correct direction. They then emphasize that Bayesian networks are conditional independencies, not dependencies, and that the arrows represent the decomposition of probabilities. Lastly, the speaker mentions that the three equations covered in the lecture, including Bayes' rule, are important for working out the probability of different events.

1.8.18 Inference By Hand Question

This lecture discusses the concept of Bayesian inference and its connection to Bayes net inference. The lecturer provides a setup where there are two boxes, one with 4 balls and the other with 5 balls. One box is chosen at random and a ball is drawn from it, which is revealed to be green. The probability of drawing a blue ball from the same box on the second draw, given that the first draw was green, is then considered. The lecturer asks a student to draw a Bayes net to represent this problem, and they explain that the variables in the net are the box variable and the color of the ball variable. Conditional probability tables are used to represent the probabilities of different outcomes for each variable. The net includes arrows to show the dependencies between the variables. The lecturer provides an example of a piece of the conditional probability table for the second ball's color given a green first ball draw from each box. The lecture concludes by stating the question that needs to be answered: the probability of the second draw being blue given that the first draw was green.

1.8.19 Inference By Hand Solution

Summary: In this lecture, the speaker discusses how to use Bayes nets to compute probabilities in a given scenario. The speaker breaks down the process step-by-step and emphasizes the importance of using the provided tables to simplify calculations. They explain that the goal is to compute the probability of the second ball being blue given that the first ball is green. To do this, they use Bayes' rule and apply the marginalization rule and the chain rule to split the problem into conditional probabilities. They then reference the table to determine the probabilities of the second ball being blue in box one and box two. They also calculate the probabilities of being in box one and box two using Bayes' rule. Eventually, they find that the probability of the second ball being blue given that the first ball is green is $6/23$. The speaker concludes by humorously expressing the desire for an algorithm to automate this process.

1.8.20 Naive Bayes

The lecture discusses the concept of Naive Bayes in the context of spam detection. The speaker explains how to draw arrows in a Bayesian network to represent the dependencies between variables such as spam, words like Viagra and Prince, and the word Udacity. The probabilities of these variables occurring in spam and non-spam messages are discussed. The speaker introduces the concept of conditional independence and how it simplifies the calculation of joint probabilities. The speaker demonstrates the use of Bayes' rule to calculate the probability of spam given certain attributes. The lecture concludes by showing the general form of the Naive Bayes formula and discussing its use in classification. The speaker also mentions the ability to infer the class given observed attribute values. The lecture ends with a humorous remark about the surprise of finding a palindrome in the context of spam classification.

1.8.21 Why Naive Bayes Is Cool

Naive Bayes is a powerful and efficient method for inference and classification. It is especially useful because the number of parameters needed to estimate probabilities is linear, rather than exponential as with other methods. The probabilities can be easily estimated by counting occurrences in labeled data. Naive Bayes is widely used, even by Google, and is successful at classification tasks. However, the "naive" assumption of conditional independence between attributes given the label is often not true, which can undermine its efficacy for Bayesian inference. Nevertheless, Naive Bayes can still produce correct labels even when the probabilities are incorrect. Another issue is the problem of zero probabilities, which can be addressed by smoothing the probabilities. Smoothing introduces an inductive bias that assumes all possibilities are at least mildly plausible. Overall, Naive Bayes is a useful and effective algorithm.

1.8.22 Wrapping Up

In this lecture, the speaker discusses the final concepts related to Bayesian inference and Bayesian networks. They recap the topics covered, including the Bayesian Network representation of joint probability distributions, inference computation, sampling, and Naive Bayes. The speaker emphasizes that exact and approximate inference in Bayesian networks is challenging. However, they highlight the usefulness of Naive Bayes in classification tasks, as it assumes independence between attributes. They also explain how Bayesian learning provides a way to determine the right hypothesis and classification by computing

probabilities. The speaker appreciates that inference in Naive Bayes allows for tractable classification and connects classification to the broader concept of Bayesian inference. They mention that inference is not limited to classification, but enables a wider range of tasks. Missing attributes are handled well in Naive Bayes due to the probabilistic nature of the model. The speaker concludes the discussion on Bayesian inference, noting that it also wraps up the topic of classification and regression in supervised learning. However, they acknowledge that the field of supervised learning is vast and there is always more to learn. They jokingly mention that the upcoming exam will provide a real-life example of supervised learning. The lecture concludes with both speakers expressing gratitude and anticipation for the next part of the course.