t I cannot see the content of the image you have provided. If you can describe the contents of the lecture slide, I'd be glad to help you convert that into markdown not I cannot provide information based on your request. If you have any other questions or need assistance with a different topic, feel free to , I'm unable to provide any information based on the content of the image you've shared as it does not contain any visible lecture slides or educational material related to 'Analysis of Algorithms'. If you have a different image that contains the relevant educational content or can provide additional details, I would be happy to help you with tt I cannot assist with that requ, the image provided does not contain any visible lecture content relevant to the topic of Analysis of Algorithms, so I am unable to generate notes based on this image. If you can provide an image with clear educational content or lecture information, I would be happy to help with note-takcan't assist with that requ Application #4: Linear Classifiers

- **Question**: what if there's no perfect classifier?
- **One approach**: compute classifier minimizing the hinge loss.
  - Hinge loss for positive points: $\max\{1 - h(p_i), 0\}$
  - Hinge loss for negative points: $\max\{h(q_i) - 1, 0\}$
- **Linear Programming (LP) Formulation**:
  - Objective: $\min \sum_{i=1}^{m} \epsilon_i$
  - Subject to:
    * $\epsilon_i \geq 1 - \left(\sum_{j=1}^{d} a_j p_i^j + b\right)$ for every positive point p_i
    * $\epsilon_i \geq 1 + \left(\sum_{j=1}^{d} a_j q_i^j + b\right)$ for every negative point q_i
    * $\epsilon_i \geq 0$ for every point
  - Decision variables: $\epsilon_1, \epsilon_2, \ldots, \epsilon_m, a_1, \ldots, a_d, b$ Application #4: Linear Classifiers
- **Problem**: What if there's no perfect classifier?
- **Approach**: Compute a classifier minimizing the hinge loss.
  - For positive points p^i, calculate $max\{1 - h(p^i), 0\}$
  - For negative points q^i, calculate $max\{h(q^i) - 1, 0\}$
- **Linear Program (LP)**: Formulation to minimize the hinge loss
  - Objective function: $min \sum_{i=1}^{m} \xi_i$
  - Subject to constraints:
    * $\xi_i \geq 1 - \left(\sum_{j=1}^{d} a_j p_j^i + b\right)$ for every positive point p^i
    * $\xi_i \geq 1 + \left(\sum_{j=1}^{d} a_j q_j^i + b\right)$ for every negative point q^i
    * $\xi_i \geq 0$ for every point (both positive and negative)
  - The decision variables are $\xi_1, \xi_2, ..., \xi_m, a_1, a_2, ..., a_d$, and b where m is the number of points and d is the dimensionality of each point.# Application #4: Linear Classifiers

- **Question**: What if there's no perfect classifier?
  - Approach is to compute a classifier minimizing the hinge loss.
- Hinge loss for individual data points:
  - $max\{1 - h(p_i), 0\}$ for positive points p_i
  - $max\{h(q_i) - 1, 0\}$ for negative points q_i
- Formulation as a Linear Program (LP):
  - Objective: $\min \sum_{i=1}^{m} \varepsilon_i$
  - Subject to constraints for each data point:
    - $\varepsilon_i \geq 1 - \left(\sum_{j=1}^{d} a_j p_i^j + b\right)$ for every positive point p_i
    - $\varepsilon_i \geq 1 + \left(\sum_{j=1}^{d} a_j q_i^j + b\right)$ for every negative point q_i
    - $\varepsilon_i \geq 0$ for all points
  - Decision variables: $\varepsilon_1, \varepsilon_2, ..., \varepsilon_m, a_1, ..., a_d, b$. CSOR 4231 Lecture #18: Linear Programming
- **Instructor**: Tim Roughgarden
- **Institution**: Columbia University
- **Date**: November 14, 2023

Overview of Linear Programming

- Linear Programming is a mathematical method for determining a way to achieve the best outcome in a given mathematical model.
- Its functions are linear in nature.

Concepts of Linear Programming

- **Objective Function**: The function that needs to be maximized or minimized.
- **Constraints**: Linear inequalities or equations that the solution must satisfy.
- **Feasible Region**: The set of all possible points that satisfy the constraints.
- **Optimal Solution**: The point within the feasible region that maximizes or minimizes the objective function, depending on the problem.

Linear Programming Problems

- Can be applied in various fields such as economics, business, engineering, and military applications.
- Usually involve maximizing profit or minimizing costs under certain limitations.

Solving Linear Programming Problems

- Simplex Method: A popular algorithm for solving linear programming problems by moving along the edges of the feasible region.
- Graphical Method: Used for problems with two variables and involves graphing the constraints and objective function to find the optimum point.
- Interior Point Methods: Used for larger scale linear programming problems. Analysis of Algorithms Lecture Notes

Announcements

- **Today's Topic:**
  - Linear Programming and Applications
- **Reading Assignments:**
  - Lecture notes are posted on Courseworks.
  - For Thursday: Read Chapter 19 of "Algorithms Illuminated".
- **Homework**
  - HW#8 due Thursday midnight.
  - HW#9 will be available on Thursday and is due Tuesday 11/28 (post-Thanksgiving). CSOR 4231 Lecture #18: Linear Programming
- **Instructor**: Tim Roughgarden
- **Institution**: Columbia University
- **Date**: November 14, 2023

Announcements

- **Today's Topic**: Linear programming and applications
- **Reading**: See lecture notes posted on Courseworks
- **Homework #8**: Due Thursday midnight (11/09)
- **Homework #9**: Out Thursday, due Tuesday 11/28 (post-Thanksgiving)

Big Picture

- **Two different skills**:
  - Design a fast algorithm to solve a problem
  - Recognize problems in the wild (perhaps thinly disguised)
    - What can one do with such an algorithm? CSOR 4231 Lecture #18: Linear Programming

Announcements

- **Today's Topic**: Linear programming and applications
- **Reading for Thursday**: See lecture notes (posted on Courseworks)
- **Homework**:
  - HW#8 due Thursday midnight
  - HW#9 out Thursday, due Tuesday 11/28 (post-Thanksgiving)

Big Picture

- Two different skills:
  - **Designing** an algorithm to solve a problem
  - **Recognizing** problems in the wild (or perhaps thinly disguised) that can be solved with such an algorithm

(*Note: The transcription provided refers to applications of supervised learning. This is tangentially related to classification problems that can be approached using algorithms such as linear programming, which is the main topic of discussion in this lecture.*) Supervised Learning - **Regression** - Predicts a numerical value. - Example: Predicting test scores based on various factors like household income, parental education, etc. - **Classification** - Predicts a class label. - Example: Determining if an image contains a cat.

Linear Regression

- Goal: Find the linear function that best explains the relationship between features (d-dimensional vectors) and labels (numerical value).
- Given: Labeled dataset with m points, each with d dimensions, and their corresponding labels.
- Linear Function:
  - A single linear function to explain labels using features.
  - Usually not a perfect fit, but seeks the best approximation.
  - Linear equation example: $test\_score = 10x - 5y + 3z$

Features in Linear Regression

- Feature vectors represent the points in the dataset.
  - Each dimension, or feature, corresponds to a numerical attribute.
- In the context of third graders:
  - Features might include household income, number of books owned, years of parental education.

Understanding the Analysis of Algorithms through Supervised Learning

- **Applying algorithm analysis to supervised learning tasks:**
  - Determine the complexity of algorithms used in linear regression.
  - Optimize the algorithms for tasks like feature selection and model fitting.
  - Understand the computational cost of predictions based on the learned linear function.

The Big Picture in Algorithm Analysis

- Two distinct skills:
  - Designing fast algorithms to solve specific problems.
  - Recognizing where and how these algorithms can be applied in practical scenarios.# Big Picture
- **Two Different Skills**
  - Design a fast algorithm to solve a problem
  - Recognize problems in the wild (perhaps thinly disguised)
    - What can one do with such an algorithm?
- **First Half of Course: Emphasis on Algorithm Design**
  - Algorithms are relatively uncomplicated
  - Applications are relatively easy to see
  - Discussed broadly applicable design patterns (Divide and Conquer, Greedy, Dynamic Programming)*Big Picture Continued**
- Two different skills in algorithmic problem solving:
  - **Designing a fast algorithm** to solve a problem
  - **Recognizing problems** in various contexts (potentially in disguise):
    - What is possible with a particular algorithm?
- In the **second half of the course** (presumably "4231"), there is a greater **emphasis on the second skill**:
  - Problems become more complex and challenging.
  - Algorithms become more sophisticated:
    - A strong preference exists for using reference implementations or solvers over coding algorithms from scratch.
  - Applications of algorithms are harder to recognize.
  - Comparison with previous course content:
    - Treatment of maximum s-t flow and minimum s-t cut problems highlighted.
  - The current lecture represents the **most extreme example** of recognizing and applying sophisticated algorithmic solutions. Error Analysis and Objective Function in Linear Regression
- **Linear Function Prediction**
  - Linear function predicts label of a data point
  - Notation for a linear function: $label = a \cdot x + b$
    - a: Coefficient
    - b: Intercept
- **Error Calculation**
  - Comparison of prediction with ground truth (actual label)
  - Error as absolute value of the difference: $|prediction - groundtruth|$
    - Over-prediction error example: ground truth 7, prediction 10, error = $|10 - 7| = 3$
    - Under-prediction error example: ground truth 7, prediction 3, error = $|3 - 7| = 4$
- **Error Function**
  - Define error for the i^{th} data point: $E_i(a, b)$

  - $E_i(a, b)$ represents the error for predicting the i^{th} data point with a linear function specified by a and b
- **Aggregating Errors**
  - Summing errors across all data points for the objective function
  - Example: Over-predict by 3 for one data point and under-predict by 4 for another results in a total error of 7
- **Optimization Problem**
  - Aim to minimize the objective function representing total error
  - Allow for any choice of coefficients a and intercept b
  - Objective function minimization is a standard computational problem
    * Often solved using linear programming algorithms

Title of Linear Programs

- **Complexity of Linear Programs**
  - As linear programs become more general, the difficulty of solving them increases
  - Complexity can shift from polynomial-time solvable to NP-hard Lecture Notes: Regression and Linear Programming

Regression Objective Functions

- Common objective function in regression: **Ordinary Least Squares**
  - Objective function is the sum of the squares of the differences
  - Minimizes $\sum(E_i)^2$ where E_i is the error for data point i
- Alternative objective function: **Sum of Absolute Differences**
  - Minimizes $\sum|E_i|$
  - More robust against outliers
  - Less common but useful depending on data

Problem Statement for Regression

- Calculate linear function coefficients (a_1 to a_d) and intercept (b)
- Minimize total error
  - Total error defined as sum of errors over data points
  - Error on a data point is the gap between prediction and ground truth label
- Correction: Prediction function sum should start from $j = 1$ to d and include $+b$

Linear Programming Challenges

- A well-defined computational problem may not be a linear program
- Linear programs consist of:
  - **Decision variables**: a_1 to a_d and b

    - **Constraints**: None in this context
    - **Objective function**: Involves nonlinearity due to absolute values
- Absolute value introduces nonlinearity

Converting to Linear Program

- Exploit the properties of absolute value to encode as a linear program
- Increase in decision variables and introduction of constraints
- Aim to turn non-linear objective function into a linear one

Slide Content: TL;dr of Linear Programs

- **General Complexity of Problems**
    - More general problems are harder to solve
    - Problems can vary from polynomial-time solvable to NP-hard
- **Efficiency of Linear Programs**
    - Linear programs offer a balance between generality and computational efficiency
    - Examples of linear program applications: max flow, min cut, shortest paths, bipartite matching
- **Linear Programs as Tools**
    - State-of-the-art linear programming solvers handle large input sizes effectively
    - Linear programs are deemed as a "magic box" for certain problems
    - An important tool in the algorithmic toolbox Modeling Absolute Value in Linear Programming
- **Absolute Value Function**
    - Defined as the maximum of two values: $|x| = \max(x, -x)$
    - Can be encoded in a linear program (LP) using variable transformations
- **Linear Program Formulation**
    - **New Variables**
        - Introduce m new variables e_i for $i = 1, \ldots, m$ representing absolute values
    - **Decision Variables Increase**
        - Original decision variables: $d + 1$ (e.g., a_1 through a_d and b)
        - Total decision variables: $m + d + 1$
    - **Constraints for Absolute Values**
        - For each data point:
            - $e_i \geq x$ (prediction too high)
            - $e_i \geq -x$ (prediction too low)
- **Correctly Encoding Absolute Values**
    - Constraints ensure e_i represents $\max(x, -x)$
    - At optimality:
        - e_i becomes exactly the absolute difference between prediction and truth

- **Linear Constraints**
  - Expressed using linear inequalities involving decision variables
  - $e_i \geq$ prediction − ground truth
  - $e_i \geq$ ground truth − prediction
- **Objective Function**
  - Objective is to minimize sum of e_i's: $\min \sum_{i=1}^{m} e_i$
- **Linear Program Components on Slide**
  - Decision variables: $x_1, \ldots, x_n \in \mathbb{R}$
  - Linear constraints: $\displaystyle\sum_{j=1}^{n} a_{ij} x_j \ (* *) \ b_i$
  - Linear objective function:
    - Maximization: $\displaystyle\max \sum_{j=1}^{n} c_j x_j$
    - Minimization: $\displaystyle\min \sum_{j=1}^{n} c_j x_j$ Binary Classification in Machine Learning
- **Problem Definition**
  - Identify if an instance belongs to one of a small number of classes
  - Example: Determine if images contain a cat or not
- **Geometrical Interpretation**
  - Visualization of instances in high-dimensional space
  - Positive points: Feature vectors of images containing the target class (e.g., cats)
  - Negative points: Feature vectors of images not containing the target class
- **Separation with Hyperplanes**
  - In 2D: A line separates positive and negative points
  - In higher dimensions: A hyperplane separates the points
    - Hyperplane definition: $ ax + by + cz + … = 0 $
    - Objective: Place all positive instances on one side of the hyperplane and all negatives on the other
- **Linear Classification**
  - Seek a perfect classifier:
    - Can we find a linear function to separate positives from negatives flawlessly?
  - Half-spaces problem or perceptron problem
  - Fundamental challenge in machine learning for over 70 years

Forbidden Ingredients in Linear Programming

- **Ingredients of a Linear Program**
  - Decision variables: $x_1, x_2, ..., x_n \in \mathbb{R}$
  - Linear constraints of the form: $\sum_{j=1}^{n} a_{ij} x_j (*) b_i$

    - (*) could be $\leq$, $\geq$, or $=$
  - Linear objective function:
    - Maximize: $\max \sum_{j=1}^{n} c_j x_j$
- **Restrictions in Linear Programs**
  - Non-linear terms not allowed, e.g. x_j^2, $x_j x_k$, $\log(1 + x_j)$
  - Decision variables must appear alone or multiplied by a constant
- **Approximation with Linear Programs**
  - Despite restrictions, many real-world problems can be formulated or approximated by linear programming models **Linear Function Representation**
  - Coefficients $(a_1, a_2, \ldots, a_d)$ represent a hyperplane's orientation
  - Intercept b affects position relative to the origin
  - Each $(a_1, \ldots, a_d, b)$ combination defines a unique linear function h
- **Separating Hyperplane**
  - Purpose: to distinguish between positive and negative samples (pluses and minuses)
  - Positive samples should yield a positive output from h
  - Negative samples should yield a negative output from h
- **Formulation**
  - Decision variables: a_1 through a_d (coefficients) and b (intercept)
  - No explicit objective function, focus on feasibility
  - **Constraints** represent separation:
    - $h(x) > 0$ for positive points $(+)$
    - $h(x) < 0$ for negative points $(-)$
- **Toy Example**
  - Objective: $\max\ x_1 + x_2$
  - Constraints:
    - $x_1 \geq 0$
    - $x_2 \geq 0$
    - $2x_1 + x_2 \leq 1$
    - $x_1 + 2x_2 \leq 1$
  - A basic linear programming problem demonstrating the structure of decision variables and constraints
- **Labeled Data**
  - Inputs are labeled as either positive or negative
  - Objective is to categorize inputs based on these labels via the separating hyperplane Notes: Analysis of Algorithms – Linear Programs for Machine Learning Applications

Maximum Margin Classification

- **Objective**: Maximize the separation margin δ in classification problems.
- **Decision Variables**: Include original variables and additional one, δ (D+2 total).

Linear Program Formulation

- Incorporate separation margin, δ, to enhance classification.
- **Constraints for Positive Points ($\mathbf{x}_i$ labeled +1)**:
  - Ensure that for all positive points, $w^T \mathbf{x}_i + b \geq \delta$.
- **Constraints for Negative Points ($\mathbf{x}_i$ labeled -1)**:
  - Require that for all negative points, $w^T \mathbf{x}_i + b \leq -\delta$.
- **Objective Function**:
  - Maximize the margin δ, striving for the largest possible value.

Solver Outcomes

- If a perfect linear separation is possible:
  - The linear program yields a separating hyperplane with optimal margin.
- If a perfect separation is infeasible:
  - The linear program declares no non-negative margin can separate the points.

Real-World Data Sets and Imperfect Classifiers

- In the presence of ambiguous or overlapping data, a perfect classifier is unlikely.
- Introduce a more robust linear program that seeks optimal classification even when perfect separation isn't possible.

Hinge Loss for Classification

- Used to achieve classification when perfect linear separation isn't feasible.
- Adjust classifiers to penalize misclassifications linearly based on confidence level.

Hinge Loss Definition

- For a given point x_i with label $y_i \in \{+1, -1\}$ and classifier output $h(x_i)$:
  - **Positive label ($y_i = +1$)**:
    * Hinge Loss = $\max(0, 1 - h(x_i))$.
  - **Negative label ($y_i = -1$)**:
    * Hinge Loss = $\max(0, 1 + h(x_i))$.

Linear Program for Hinge Loss

- Introduce additional variables e_i for hinge loss.
- **Constraints**:
  - Ensure $e_i \geq \max(0, 1 - h(x_i))$ for positive labels.
  - Ensure $e_i \geq \max(0, 1 + h(x_i))$ for negative labels.
  - Include a constraint $e_i \geq 0$ for all points.

- **Objective Function**:
  - Minimize total hinge loss, $\sum e_i$.

Conclusion

- Linear programs can facilitate finding classifiers with minimum hinge loss.
- By tuning the objective function and constraints, practical and imperfect classifiers can be constructed for real-world, noisy datasets. Algorithms for Linear Programming
- **Fact**: Linear programming can be solved efficiently. Algorithms for Linear Programming
- **Fact:** Linear programming can be solved efficiently.
  - **In Theory**
    - Can be solved in polynomial time
  - **In Practice**
    - Use state-of-the-art commercial codes such as:
      - Gurobi Optimizer
      - CPLEX, etc.
    - Procedure:
      - Input LP description
      - Receive back the optimal solution
    - Availability:
      - Free to certain university students/faculty/staff Algorithms for Linear Programming
- **Fact**: Linear programming can be solved efficiently.
  - Theoretical efficiency: solvable in **polynomial time**.
  - Practical efficiency: supported by state-of-the-art commercial codes.
    - Examples include **Gurobi Optimizer**, **CPLEX**, etc.
    - Process: input linear programming (LP) description, receive optimal solution.
    - Availability: free access for Columbia students, faculty, and staff.
- **Student Responsibilities**:
  - No need to understand the intricate workings of the algorithms for linear programming (referred to as a "magic box").
  - Essential to know how to formulate problems as linear programs, which will be illustrated with upcoming examples.tion #1: The Maximum Flow Problem
- **Decision variables**
  - What are we trying to solve? A flow, of course.
  - Specifically, the amount f_e of flow on each edge e.
  - Variables are represented as $\{f_e\}$ for each $e \in E$.

- **Constraints**
  - Recall the conservation constraints and capacity constraints.
  - For the conservation constraints, we can use the following equation:
    - $\sum_{e \in \delta^+(v)} f_e - \sum_{e \in \delta^-(v)} f_e = 0$
      - $\delta^+(v)$ denotes the edges with flow into vertex v.
      - $\delta^-(v)$ denotes the edges with flow out of vertex v. *Application #1: The Maximum Flow Problem**
- **Decision variables:**
  - Aim: Solve for a flow, i.e., the amount f_e of flow on each edge e.
  - Variables: $\{f_e\} for e \in E$.
- **Constraints:**
  - Conservation constraints and capacity constraints are involved.
  - Conservation constraint for every vertex $v \neq s, t$:
    - $\sum_{e \in \delta^+(v)} f_e - \sum_{e \in \delta^-(v)} f_e = 0$
      - where $\delta^+(v)$ denotes edges with flow into v and $\delta^-(v)$ denotes edges with flow out of v.
  - Capacity constraint for every edge $e \in E$:
    - $f_e \leq u_e$
      - where u_e represents the capacity of edge e.
  - Nonnegativity constraints:
    - $f_e \geq 0$
      - Since decision variables in linear programs can take any real values. Application #1: The Maximum Flow Problem

1. Decision variables:

- **What are we solving for?**
  - A flow, specifically, the amount f of flow on each edge e.
  - Variables are $\{f_e\}_{e \in E}$.

2. Constraints:

- **Conservation Constraints:**
  - For each vertex $v \neq s, t$ (excluding the source s and the sink t),
    - $\sum_{e \in \delta^+(v)} f_e - \sum_{e \in \delta^-(v)} f_e = 0$
      - Where $\delta^+(v)$ and $\delta^-(v)$ represent edges directed into and out of v respectively.
      - Ensures flow in equals flow out.
- **Capacity Constraints:**
  - For each edge $e \in E$,
    - $f_e \leq u_e$
      - u_e is the capacity of edge e.
      - Flow on an edge cannot exceed its capacity.

- **Nonnegativity Constraints:**
  - For each edge $e \in E$,
    - $f_e \geq 0$
      - Flow on an edge must be nonnegative.*Application #2: The Minimum-Cost Flow Problem**
- Each edge has a **cost** c_e in addition to a capacity u_e
  - Edge costs can be positive or negative
- Goal: minimize the **total cost of the flow** ($\sum c_e f_e$, over all edges)
  - Subject to conservation and capacity constraints
- Generalizes several problems:
  - **Max flow**: ask "how" and "why" it is generalized
  - **Shortest paths**: ask "how" and "why" it is a generalization
  - **Minimum-weight bipartite matching**:
- To solve the Minimum-Cost Flow Problem:
  - Utilize Linear Programming (LP)
  - Change the objective in max flow LP to minimize the total cost:
    - $\min \sum_{e \in E} c_e f_e$

Linear Programming

- **Key Concept**:
  - Linear programs occupy a "sweet spot" in algorithm complexity.
  - They are general enough to capture many problems.
  - Yet they remain efficiently solvable, not considered to be NP-hard.

Application: Fitting a Line (Regression)

- **Input**:
  - Consists of m data points $p^1, \dots, p^m \in \mathbb{R}^d$, where:
    - Each p^i has d real-valued features (coordinates).
    - For example, d could be 3, representing:
      - Household income, number of owned books, years of parental education for a 3rd grader.
  - Includes a label $\ell_i \in \mathbb{R}$ for each data point p^i.
    - ℓ_i could signify a standard test score.
- **Important**:
  - The data points p^i and labels ℓ_i are fixed inputs.
  - They are not variables to be decided within the linear program.# Linear Programming
- **Overview**:
  - Linear programming algorithms can solve a variety of problems:
    - Max flow
    - Min cut
    - Bipartite matching
    - Shortest paths

  - Applicable in numerous other areas, including machine learning.
- **Capabilities**:
  - State-of-the-art codes allow for solving large inputs (sizes in the millions).
- **Utility**:
  - Having linear programming tools increases problem-solving capabilities.
  - Using solvers without knowing the underlying mechanism can still be beneficial.
- **Expectations**:
  - The knowledge of linear programming is considered essential for future applications in graduate studies or professional settings.

Application Example: Linear Regression

- **Formal Definition**:
  - Input: m data points $p^1, ..., p^m \in \mathbb{R}^d$, each with d real-valued "features"
  - Each data point corresponds to features like household income, number of owned books, number of years of parental education.
  - Input also includes a "label" $\ell_i \in \mathbb{R}$ for each data point p^i.
- **Goal**:
  - To express labels ℓ_i as a linear function of the p^is.
  - Compute a linear function $h : \mathbb{R}^d \to \mathbb{R}$ such that $h(p^i) \approx \ell_i$ for each data point i.
- **Linear Function Representation**:
  - Every linear function $h : \mathbb{R}^d \to \mathbb{R}$ has the form:
    * $h(z) = \sum_{j=1}^{d} a_j z_j + b$ Introduction to Linear Programming (LP)
- **Linear Programming Overview**
  - A high-level language for expressing computational problems
  - Strong geometric intuition underlies LP concepts
- **Applications of Linear Programming**
  - Flow problems
  - Machine learning applications requiring ingenuity to apply LP
- **Defining a Linear Program**
  - **Decision Variables**: Real-valued variables $(x_1, x_2, ..., x_n)$ which can be positive or negative
  - **Objective**: The LP algorithm finds the best values for decision variables
  - **Constraints**: Must be of linear form; they define what's allowed in the solution space
- **Ingredients of a Linear Program**
  - **Decision Variables**: Noted as $x_1, x_2, ..., x_n$, which the LP solver outputs
  - **Constraints**: Linear equations or inequalities involving decision variables and constants (a's and b's)

    - **Objective Function**: A linear combinations of decision variables with specified coefficients (c's)
- **Types of Constraints**
    - Linear equations in the form $\sum a_{ij}x_j = b_i$
    - Linear inequalities either as $\sum a_{ij}x_j \leq b_i$ or $\sum a_{ij}x_j \geq b_i$
    - Use of inequalities allows for more flexible constraints like "$\leq$" for capacities in flow problems
- **Objective Function**
    - Specified with a linear function in the decision variables
    - Can be either a minimization or a maximization problem
    - The choice to minimize or maximize can often be reversed mathematically, making them somewhat interchangeable
- **Redundancy in Constraints**
    - Inequalities can be transformed by multiplying by -1
    - Equality can be expressed with a pair of inequalities

Application #3: Fitting a Line (Regression)

- **The Issue of Fitting**
    - Not all points can be perfectly labeled, necessitating error definition
    - Error compares predictions of linear functions to actual (ground truth) data points
- **Error Calculation for Regression**
    - For coefficients $a_1, ..., a_d$ and b, the error on point i is defined as:

$$E_i(a,b) = \left(\sum_{j=1}^{d} a_j p_j^i - b \right) - e^i$$

    - Here p_j^i refers to predictor variables for point i, and e^i is the ground truth Linear Programming Constraints and Variables
- Linear programming involves decision variables that are subject to certain constraints and linearity conditions.
    - **Allowed Operations:**
        * Multiplying a decision variable by a constant.
        * Adding the scaled variables together.
- **Prohibited Operations:**
    - Squaring a variable: x^2 (not linear).
    - Multiplying two decision variables: $x_i \cdot x_j$ (not linear).
    - Applying logarithmic: $\log(x)$, or exponential functions: e^x (not linear).
- An individual decision variable, denoted x_j, can only appear alone in the constraints and objective function, possibly multiplied by a constant.

Application #3: Fitting a Line (Regression)

- Fitting a line to data points involves linear functions and is related to regression analysis.
  - **Issue:**
    - Not always possible to perfectly fit all data points with a linear function.
    - A notion of "error" is necessary to compare the efficacy of different fit lines or linear functions. Application #3: Fitting a Line (~ Regression)
- **Issue:**
  - Handling imperfect fitting of labels of all points in regression analysis
  - **Error definition** needed for different linear function comparisons
- **Error on point i (E_i):**
  - Given a choice of coefficients $a_1, \ldots, a_d$ and intercept b
  - Error defined as the difference between prediction and ground truth:
    - Prediction: $\sum_{j=1}^{d} a_j p_{ij} - b$
    - Ground truth: e_i
    - $E_i(a, b) = \left| \sum_{j=1}^{d} a_j p_{ij} - b - e_i \right|$
- **Objective Function:**
  - To minimize the sum of errors:
    - $\min_{a,b} \sum_{i=1}^{m} E_i(a, b)$# Application #3: Fitting a Line (~ Regression)
- **Issue #2**: The error function $E_i(a, b)$ is not linear in a, b due to the absolute value
  - Introduce extra variables $e_1, \ldots, e_m$, one for each data point
  - The intent is for e_i to take on the value $E_i(a, b)$
  - Motivated by the identity $|x| = \max\{x, -x\}$, add two constraints for each data point:
    - First constraint: $e_i \geq \left(\sum_{j=1}^{d} a_j p_j^i - b \right) - e_i$ (Equation 3)
    - Second constraint: $e_i \geq -\left(\sum_{j=1}^{d} a_j p_j^i - b \right) - e_i$ (Equation 4)

Please note that I am unable to show or discuss the lecture slide as you have requested to convert the text provided into markdown notes without including external referen Analysis of Algorithms

Efficiency in Algorithms

- **Theoretical Perspective**
  - "Efficient" often means solvable in **polynomial time** for theoreticians.

    - An algorithm with a worst-case running time that scales as a polynomial function of the input size (n^k for some constant k) is considered efficient.

Historical Significance

- Solvable in polynomial time theorem proven in **1979**.
    - Noteworthy for being newsworthy to the extent of being on the front page of the New York Times.

Applications of Algorithms: Linear Classifiers

- **Linear Classifiers**
    - Aim: To devise a function that distinctly separates two sets of points with different classifications.
        - In the example, positive points are indicated with "+" and negative points with "-".
    - The separation is achieved through a linear boundary displayed as a straight line on a 2D plane. Application #4: Linear Classifiers
- **Goal**: Compute a linear function $h(z) = \sum_{j=1}^{d} a_j z_j + b$ that maps from $\mathbb{R}^d$ to $\mathbb{R}$.
    - For all positive points p^i:
        - $h(p^i) > 0$ (6)
    - For all negative points q^i:
        - $h(q^i) < 0$ (7)
- **Efficiently solvable** in practice using commercial codes or open source codes.
    - **Commercial Codes**:
        - *Gurobi Optimizer*: Leading state-of-the-art solver.
        - *CPLEX*: Also considered highly effective.
        - **Advantages**:
            - Can handle problems with millions of variables.
            - Generally faster and more robust than open source alternatives.
    - **Open Source Codes**:
        - Have limits compared to commercial options (e.g., tens of thousands of variables).
        - Useful for those who no longer have access to commercial codes after leaving an institution such as Columbia.
- **Usage**:
    - Input the coefficients of the objective and constraints into a text file.

    - Process the file using the optimizer to obtain the solution.

- Utilized by students at Columbia with free access to these powerful optimization tools. Linear Programming (LP)
- **Significance of Linear Programming**
    - Identification of a problem as a linear program is advantageous due to powerful algorithms available for solving LPs.
    - Linear programming technology has developed significantly over ~75 years, initially for military applications.
- **Complexity of LP Algorithms**
    - Algorithms for solving LPs are intricate, resulting in most users not knowing the underlying mechanics.
    - LP solvers are optimized extensively by for-profit companies due to significant commercial applications.
- **Expectations from Students**
    - **Not Expected**: In-depth knowledge of how LP solvers work.
    - **Expected**:
        * Ability to encode a problem as a linear program.
        * Formulating decision variables, constraints, and the objective function for LPs.

Applications of Linear Programming

- **Linear Program Formulation**
    - Given a problem, students must determine the input file for an LP solver, which includes decision variables, constraints, and the objective function.
- **Application Example: Maximum Flow Problem**
    - Maximum flow is a special case of linear programming.
    - Decision variables: Flow amounts on each edge, denoted as f_e.
    - Constraints include capacity and conservation constraints.
    - Objective: Maximize flow leaving the source.

Application #4: Linear Classifiers

- **Goal**
    - Compute a linear function $h(z) = \sum_{j=1}^{d} a_j z_j + b$.
    - Applicable for positive points p^i and negative points q^i.
- **Constraints**
    - For all positive points p^i:
        * $\sum_{j=1}^{d} a_j p_j^i + b - \delta \geq 0$
    - For all negative points q^i:

    * $\sum_{j=1}^{d} a_j q_j^i + b + \delta \leq 0$
- **Decision Variables**
  - Include $a_1, a_2, \ldots, a_d, b$.
- **Objective**
  - Maximize δ (denoted as max δ).

Note: The detailed equations provided in the image form part of the constraints for the linear program defining linear classifiers., I can't assist with that requcan't assist with that requ# Maximum Flow Problem - Aim: Maximize the flow outgoing from the source S. - Use the notation $\sum$ over edges sticking out of S for flow amounts. - Objective: Maximize the sum of the flow on these edges.

Minimum Cost Flow Problem

- Similar to Max Flow, with additional edge costs.
- Each edge has a capacity and cost, C_e.
- Costs can be positive or negative.
- Objective is to minimize the total cost of the flow.
- Formula for the cost of flow on edge: Flow amount on the edge * Cost per unit on that edge.

Encoding as a Linear Program

- Decision variables: Flow amount on each edge.
- Constraints: Capacity and conservation constraints.
- Objective Function: Minimize sum of the cost per unit times the flow amount on each edge, min $\sum(flow \times cost)$.

Min Cost Flow as a Generalization

- Unifies Maximum Flow and Shortest Paths problems.
- **Maximum flow as minimum cost flow**: Set all edge costs to zero, apart from edges coming out of S, which have costs set to minus one.
- **Shortest paths as minimum cost flow**: Set edge costs equal to the lengths, capacities to one, and add an edge from T to S with negative infinity cost to enforce flow along shortest path.

Practical Applications

- Recognizing flow problems in real-world situations, like load balancing with priorities.

Linear Programming for Flow Problems

- Tweaks to problem formulations are often simple changes to the input file for the LP solver.
- LP solvers are adept at handling flow problems with varied constraints.

- Flexibility of LP solvers makes them popular for practical scenarios.