CS 165B Notes

Alex Mei

Fall 2021

1 Introduction

- **Machine Learning**: build computer systems that learn from data that are capable of adapting to environments.
- **Discriminating Tasks**: identifying (decision rules) from given data. (i.e., decision trees.)
- **Generative Tasks**: create new data/statistical models. (i.e., Bayesian networks.)
- **Instance Based Tasks**: uses observations instead of models. (i.e., K-Nearest Neighbors.)

2 Linear Algebra

2.1 Matrix Multiplication

- **Definition:** given $A \in \mathbb{R}^{m \times p}, B \in \mathbb{R}^{p \times n}, C = AB = B \in \mathbb{R}^{m \times n}$ where

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}$$

- **Computational Complexity**: $O(mpn)$ because there are $m \times n$ inner products of length p.

2.2 Norms

- L_1 **Norm**:

$$||x||_1 = \sum_{i=1}^{n} |x_i|$$

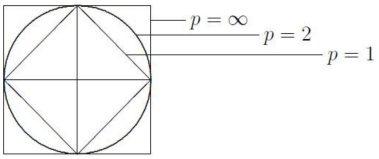
- L_2 **Norm**:

$$||x||_2 = \sqrt{\sum_{i=1}^{n} |x_i^2|}$$

- L_∞ **Norm**:

$$||x||_\infty = \max_{1 \leq i \leq n} |x_i|$$

- **Norm Balls**: every point on the norm ball is equal value for that norm.



- **Frobenius Norm (for Matrices)**:

$$||A||_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}^2|}$$

- **Trace**: given $B \in \mathbb{R}^{n \times n}$:

$$trace(B) = \sum_{i=1}^{n} b_{ii}$$

- Theorem: $||A||_F^2 = trace(AA^T)$.

2.2.1 Norm Properties

- Vector norm is a mapping from $\mathbb{R}^n \to \mathbb{R}$.
- $\forall_x ||x|| \geq 0$
- $||x|| = 0 \iff x = 0$
- $||ax|| = |a|||x||, a \in \mathbb{R}$
- $\forall_{x,y} ||x + y|| \leq ||x|| + ||y||$

2.2.2 Vector Similarity

- **Cosine**: normalized dot product; closer vectors approach $\cos\theta = 1$; distant vectors approach $\cos\theta = 0$.

$$\cos\theta = \frac{q^T d}{||q||_2 ||d||_2}$$

- **Euclidean**: affected by scale.

$$||.|| = \sqrt{\sum_i (q_i - d_i)^2}$$

2.3 Linear Independence

- **Definition**: $\sum_{i=1} \alpha_i v_i = 0$ for vectors $\{v_1, ..., v_n\}$ iff $\forall_{i \in (1,n)} \alpha_i = 0$.
- **Linear Combination**: a vector that is a linear sum of scaled vectors $\{v_1, ..., v_n\}$.
- Note: if a set of vectors are linearly dependent, then one of them can be written as a linear combination of the others.
- **Basis**: A set of m linearly independent vectors of $\mathbb{R}^m$ such that any vector in $\mathbb{R}^m$ can be expressed as a linear combination of the basis vectors.
- **Rank**: maximum number of linearly independent column vectors.
- **Singular**: a matrix with determinant 0.
- Note: Non-singular matrices have full rank.
- **Inverse**: all nonsingular matrices have an inverse matrix satisfying

$$AA^{-1} = A^{-1}A = I$$

2.4 Orthogonality

- **Definition**: Two vectors x, y are orthogonal, if $x^T y = 0$.
- Theorem: a set of orthogonal vectors $\{v_1, ..., v_n\} \in \mathbb{R}^m$ with $m \geq n$ are linearly independent.
- **Orthonormal Basis**: a set of n normalized orthogonal vectors that span $\mathbb{R}^n$.
- **Orthogonal Matrix**: matrix with orthonormal columns.

2.4.1 Orthogonal Matrix Properties

- An orthogonal matrix $Q \in \mathbb{R}^{m \times m}$ has rank m.
- $Q^{-1} = Q^T$.
- $||Qx||_2^2 = ||x||_2^2$
- If Q, P are orthogonal matrices, QP, PQ are orthogonal matrices.

2.5 Eigendecomposition

- **Eigenvalues λ and Eigenvectors** v: for any nonzero vector v

$$Av = \lambda v$$

- To find λ, v, solve:

$$det(A - \lambda I) = 0$$

- Note: since there are infinite v, we want to normalize v.
- Theorem: if λ is an eigenvalue of A with eigenvector v, then if $k > 0$, λ^k is an eigenvalue of A^k with eigenvector v.
- Theorem: if P is an invertible matrix, A and $P^{-1}AP$ contain the same set of eigenvalues.
- Property: $det(AB) = det(A)det(B) = det(B)det(A)$.
- **Eigendecomposition**: a symmetric matrix A can be written as $U\Lambda U^T$ where U is an orthogonal matrix containing the the eigenvectors of A and Λ is a diagonal matrix with the associated eigenvalues.

2.5.1 Properties

- If A is symmetric, its eigenvalues are unique and real.
- If A is symmetric, its eigenvectors are mutually orthogonal.

2.6 Positive Definiteness

- **Positive Semi Definite**: a matrix A that satisfies $x^T Ax \geq 0$ for all vectors x.
- **Positive Definite**: a matrix A that satisfies $x^T Ax > 0$ for all nonzero vectors x.

2.6.1 Properties

- All eigenvalues of A are nonnegative (PSD) or positive (PD).
- $X^T AX$ for any matrix X is PSD if A is PSD or PD if A is PD.
- All diagonal entries of a PD matrix A are positive.

2.7 Matrix Derivative

- **Gradient**: shape stays the same

$$\nabla_w f(w) = \begin{bmatrix} \frac{\delta}{\delta w_1 f(w)} \\ ... \\ \frac{\delta}{\delta w_n f(w)} \end{bmatrix}$$

- **Jacobian Matrix**: $f : \mathbb{R}^n \to \mathbb{R}^m, J(f)$ outputs $\mathbb{R}^{m \times n}$

$$J(f) = \begin{bmatrix} \frac{\delta f_1}{\delta w_1} & ... & \frac{\delta f_1}{\delta w_n} \\ ... & ... & ... \\ \frac{\delta f_m}{\delta w_1} & ... & \frac{\delta f_m}{\delta w_n} \end{bmatrix}$$

- **Hessian Matrix**:

$$H(f) = J(\nabla_w (f(w)) = \begin{bmatrix} \frac{\delta f}{\delta w_1^2} & \frac{\delta f}{\delta w_1 w_2} & ... & \frac{\delta f}{\delta w_1 w_n} \\ \frac{\delta f}{\delta w_2 w_1} & \frac{\delta f}{\delta w_2^2} & ... & \frac{\delta f}{\delta w_2 w_n} \\ ... & ... & ... & ... \\ \frac{\delta f}{\delta w_n w_1} & \frac{\delta f}{\delta w_n w_2} & ... & \frac{\delta f}{\delta w_n^2} \end{bmatrix}$$

- **Second Derivative Test**: $f''(x) > 0$ or $H(f)$ is positive definite implies local maximum. $f''(x) < 0$ or $H(f)$ is negative definite implies local minimum.

3 Perceptron

- **Input**: vector $x = [x_1, ..., x_d]^T$ with d dimensions.
- **Output**: binary class mapped from a score $= \sum_{i=0}^{d} w_i x_i$ based on some threshold.
- **Goal**: find optimal weights $w = [w_0, ..., w_d]^T$ to determine importance of features and return best result.
- Note: negative weights can denote detrimental features.
- Note: w_0 is the bias term with $x_0 = 1$.
- **Perceptron Function**: a linear/hyperplane separator

$$h(x) = sign(w^T x)$$

- **Hypothesis Set**: an uncountably infinite set of functions that makes up the perceptron classifier

$$H = \{h(x)\}$$

- Want to learn one of infinitely many $g(x) \in H$ that minimizes error for

$$\min_{error} (g(x_i) = y_i)$$

- Theorem: if data is linear separable, then PLA can find a linear separator in a finite number of steps where $y * (w^*)^T x > \delta$.
- **Cauchy Schwartz Inequality**:

$$((w^*)^T w(t))^2 \leq ||w^*||_2^2 ||w(t)||_2^2$$

- **PLA Convergence Bound**:

$$t \leq \frac{||w^*||_2^2 \max_i ||x_i||_2^2}{\delta^2}$$

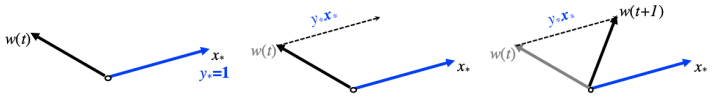
- δ measures how close the decision boundary is to the input patterns.
- Data with a smaller δ, has a higher bound needed for convergence.

3.1 Perceptron Learning Algorithm (PLA)

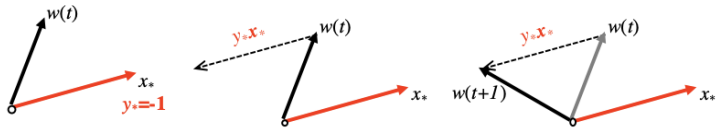
- Initialize $w(t=1)=1$
- Repeat for a fixed number of iterations t_{max} or until converged:
- Pick a random misclassified example (x_*, y_*).
- Update weight $w(t+1) = w(t) + y_* x_*$.

3.1.1 Geometric Interpretation

- **Positive Example**: y_*, x_* in same direction. $y_* = 1, g(x) = -1$, meaning angle between $x_*, w > 90°$. We update to decrease angle size:



- **Negative Example**: y_*, x_* in opposite direction. $y_* = 1, g(x) = 1$, meaning angle between $x_*, w < 90°$. We update to increase angle size:



3.2 PLA Issues

- Cannot fit nonlinear data.
- Assumes training and production environment have same distribution of data.

4 Probability

- **Uncertainty**: arises both through noise on measurements and the finite size of data sets.
- Probability theory provides a consistent framework for the quantification and manipulation of uncertainty.
- When combined with decision theory, it allows us to make optimal predictions given all the information available to us, even though that information may be incomplete or ambiguous.
- **Event**: set of outcomes of a random process.

$$0 \leq P(A) \leq 1$$

- **Sample Space**: set of all possible outcomes.

$$P(S) = 1$$

- **Complement**: given event A, the complement is the event that A does not occur.

$$P(A') = 1 - P(A)$$

- **Union**: the event that either events A or B or both occurs.

$$P(A \cup B) = P(A) + P(B) – P(A \cap B)$$

- **Intersection**: the event that both events A and B occur.
- **Disjoint**: two events that cannot happen at the same time.

$$P(A \cup B) = P(A) + P(B)$$

- **Independent**: two events that do not affect the probability of one another.

$$P(A \cap B) = P(A) * P(B)$$

- **Random Variable**: numerical outcome of an event.
- **Prior (Unconditional) Probability**: probabilities associated with a proposition or variable, prior to any evidence.
- **Posterior (Conditional) Probability**: probabilities of query variable(s) after evidence variable(s) is/are gathered:

$$P(X|Y) * P(Y) = P(X, Y)$$

- **Expectation**:

$$E(X) = \int_X x * P(x)$$

- **Moment**:

$$E(X^n) = \int_X x^n * P(x)$$

- **Variance**:

$$V(X) = E(X^2) - [E(X)]^2$$

- **Standard Deviation**:

$$\sigma(x) = \sqrt{V(X)}$$

- **Marginalization**:

$$P(X) = \int_Y P(X, Y) = \int_Y P(X|Y)P(Y)$$

- **Joint Probabilities**:

$$P(X_1) = \int_{X_2} ... \int_{X_n} P(X_1, ...X_n)$$

- **Conditional Independence**: two events A, B are conditionally independent on event C iff

$$P(A|B, C) = P(A|C)$$

- Note: independence does not imply conditional independence.
- **Gaussian PDF**:

$$f(x) = \frac{\exp(-\frac{(x-\mu)^2}{2})}{\sigma\sqrt{2\pi}}$$

- **Maximum Posterior Estimation (MAP)**: solve for maximum likelihood parameters based on given data. Use log likelihood transformation and take derivative with respect to parameter of interest. Use second derivative test to confirm the direction of extrema.

5 Linear Regression

- **Input**: feature vector x.
- **Output**: real valued-target t.
- **Goal**: fit an M order polynomial function that is linear with respect to w

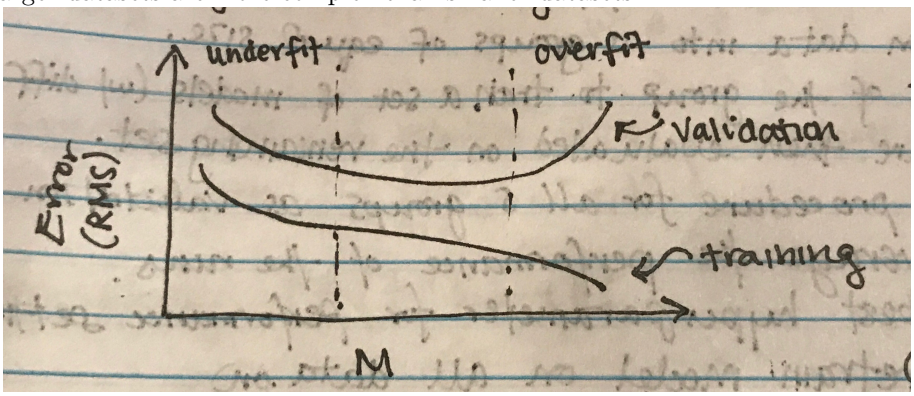
$$y(x, w) = w_0 + w_1 x + ... + w_m x^M$$

- Need to find an optimal w^* that minimizes the error function.
- **Least Squares Error**:

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}(y(x_i, w) - t_i)^2 = \frac{1}{2}||Xw - t||_2^2$$

5.1 Fitting

- **Underfitting**: high train and validation error because model is too weak.
- **Overfitting**: low training error but high validation error because model not generalizable.
- We can notice overfitting if weights are abnormally large.
- For good fit region, we want to pick the least complex function that well fits the data.
- Easier datasets are easier to overfit.
- Larger datasets are more complex than smaller datasets.



5.2 Cross Validation (s-fold)

- Partition data into s equal sized groups.
- Select one group as a validation set and use the other groups for training. Repeat for all groups and average the performance.
- Repeat process for candidate hyperparameters and select best choice.
- Retrain model on full training data.

5.3 Maximum Likelihood

- Least Squares is closely related to maximum likehihood of a dataset when there is Gaussian noise.
- Assumption: observations from a deterministic function with added Gaussian noise

$$P(t|x, w, \beta) = N(t|y(x; w), \beta^{-1})$$

- **Gaussian Noise Likelihood**:

$$\ln(P(t|x, w, \beta) = -\frac{N}{2}\ln(2\pi) + \frac{N}{2}\ln(\beta) - \frac{\beta}{2}\sum_{i=1}^{n}(t_i - w^T x_i)^2$$

- **Least Squares Problem**:

$$\min_w(\frac{1}{2}||Xw - t||_2^2) = \min_w(\frac{1}{2}(w^T X^T Xw - 2w^T X^T t + t^T t))$$

$$\nabla_w E(w) = X^T Xw - X^T t$$
$$w^* = (X^T X)^{-1} X^T t$$

- Note: we can guarantee w^* is a local minimum because $||Xw - t||_2^2$ is a convex function.
- **Predictions**:

$$\hat{y} = \tilde{X}w^*$$

- Property: $\nabla_w(w^T Aw) = (A + A^T)w$.
- Property: $\nabla_w(w^T b) = b$.

5.4 Regularization

- **Definition**: penalty term to constrain solution set and model complexity to prevent overfitting.
- **Ridge Regression Error**: λ denotes the regularization constant that determines the relative importance

$$E(w) = \frac{\lambda}{2}||w||_2^2 + \frac{1}{2}\sum_{i=1}^{n}(y(x_i, w) - t_i)^2 = \frac{\lambda}{2}||w||_2^2 + \frac{1}{2}||Xw - t||_2^2$$

$$\nabla_w E(w) = X^T Xw - X^T t + \lambda w$$
$$w^* = (X^T X + \lambda I)^{-1} X^T t$$

- **Rank Deficiency**: $(X^T X)^{-1}$ may not exist (definitely if sample size too small). Can resolve with regularization term.
- **General Regularization Framework**:

$$E(w) = \frac{\lambda}{2}||w||_q^q + \frac{1}{2}\sum_{i=1}^{n}(y(x_i, w) - t_i)^2 = \frac{\lambda}{2}||w||_q^q + \frac{1}{2}||Xw - t||_2^2$$

- **Lasso Regression**: using regularization framework $q = 1$.

6 Optimization

- **Convexity**: a convex function guarantees a single local = global minimum:

$$\forall_{\alpha\in(0,1),w_1,w_2} f(\alpha w_1 + (1-\alpha)w_2) \leq \alpha f(w_1) + (1-\alpha)f(w_2)$$

- Function value is less than the linear interpolation any two points beside it (convex case). For concave case, function value is greater than linear interpolation.
- **Optimization Step**:

$$w(t+1) = w(t+1) + \eta\hat{v}$$

- **Direction Vector**: pick $\hat{v}$ to minimize $E(w(t+1))$ direction.

$$g_t = \nabla_w E(w(t))$$

$$v_t = -g_t$$
$$\hat{v} = -\frac{g_t}{||g_t||_2}$$

- ΔE:

$$\Delta E = E(w(t)) - E(w(t+1)) \approx -\eta \nabla_w E(w(t))^T \hat{v}$$

- Need η to be small enough so we do not overshoot correction.
- Need η to be large enough to learn fast enough.
- Can adaptively change step size to increase speed and minimize overshooting.
- **Assumption**: our error function must be smooth/everywhere differentiable.

6.1 Gradient Descent Algorithm

- Initialize $w(t=0) = 0$.
- Repeat until a maximum number of iterations or until converged:
- Compute the gradient and direction and update the weights.
- Return the final weights.

6.1.1 Termination Conditions

- Calculate difference or divergence between $E(w(t+1)), E(w(t))$. (Robust version using average of last k steps.)
- Calculate $||g_t|| < \delta$.

6.2 Batched/Stochastic Gradient Descent

- Sample data points to calculate gradient to be less memory intensive.
- Redefine loss function as

$$E(w) = \frac{1}{n}\sum_{i=1}^{n} l(w, x_i, t_i)$$

- **Algorithm**: randomly selects a point (or batch) and runs the gradient descent algorithm.
- **Tradeoff**: improves efficency by introducing noise, making convergence rate longer. The average move is the same as gradient descent, but simpler, and can help escape local minimum for nonconvex functions.

7 Logistic Regression

- **Premise**: map linear function to classes using the sigmoid function to transform range to (0, 1).
- **Classifier**:

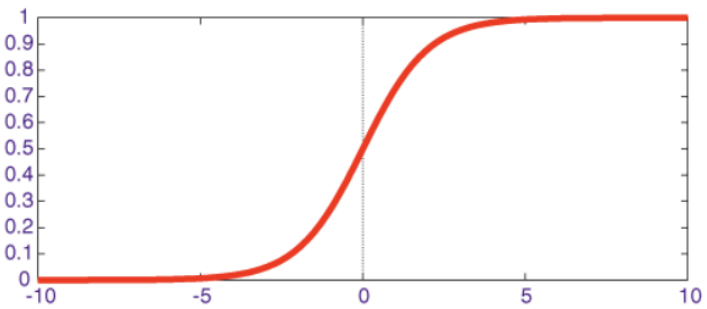
$$h(x) = \theta(w^T x)$$

- **Sigmoid Function**:

$$\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$$

- Property:

$$\theta(s) + \theta(-s) = 1$$



- **Minimizing Loss**: need to use gradient descent as there is no closed form solution.

- **Logistic Loss Function**:

$$E(w) = \frac{1}{n}\sum_{i=1}^{n} \ln(1 + \exp(-y_i * w^T x_i))$$

- Note: $y_i = 1$ encourages $w^T x_i >> 0$ such that $h(x_i) \to 1$.
- Note: $y_i = -1$ encourages $w^T x_i << 0$ such that $h(x_i) \to 0$.
- Note: this error function is equivalent to the maximum likelihood interpretation with $P(y|x) = \theta(y * w^T x)$.
- **Gradient of Logistic Loss**:

$$\nabla_w E(w(t)) = \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i * x_i}{1 + \exp(y_i * w^T x_i)}$$

- For Stochastic Gradient Descent with random sample (x_*, y_*) (helpful when dataset too large), use

$$\nabla_w E(w(t)) = \frac{-y_* * x_*}{1 + \exp(y_* * w^T x_*)}$$

- Note: even with 1 random sample, $y_* * w^T x_* \to \infty$ as (x_*, y_*) is more correctly classified, mitigating the gradient. Similarly, for PLA, w has an increasing norm, reducing the contribution of (x_*, y_*).

8 Nonlinear Transformations

- **Premise**: apply transformation function $\Phi : X \to Z$ on nonlinear data $X \in \mathbb{R}^d$ to map to linearly separable data $Z \in \mathbb{R}^{d'}$.
- **Order Polynomial Transformation**: apply a k degree transform to map data into higher dimensional data.
- For example, $(1, x_1, x_2)$ would map to $(1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)$ for a 2nd order transformation.
- Note: since dimensionality increases rapidly, this is prone to overfitting.
- **Data Snooping**: looking at data and tailoring transformations. Not possible for complex datasets and can make data hard to generalize.

9 Naive Bayes

- **Baye's Rule**:

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

- **Note**: here, $P(Y|X)$ is causal (likelihood) knowledge.
- **Bayes Parameters**: Needs $c(k^d - 1)$ independent parameters to complete the joint probability distribution for Baye's rule, where c is classes and d is features with k values each plus $c - 1$ parameters to compute $P(X)$.
- **Naive Bayes Model**: model that tries to predict a single cause given its many influence variables, which are conditionally independent given that cause.

$$c(x) = \max_{c_j \in C} P(x_1, ..., x_n|c_j) * P(c_j) = \max_{c_j \in C} P(c_j) * \Pi_{1 \le i \le n} P(x_i|c_j)$$

- **Naive Bayes Assumption**: Assume the probability of observing $P(x_1, ..., x_n|c_j)$ is equal the product of $P(x_i|c_j)$.
- **Naive Bayes Parameters**: Needs $c * d(k - 1)$ independent parameters to complete the joint probability distribution for Baye's rule, where c is classes and d is features with k values each plus $c - 1$ parameters to compute $P(X)$.
- **Text-based Naive Bayes**: represent feature vector with 1 or 0 based on whether the word appears in a given text. Then, use 1 or 0 as the possible value of a feature (word). Compute the highest probability class.

9.1 Numerical Data for Naive Bayes

- Can model the probability as the normal pdf given the sample mean and sample variance.
- **Sample Mean**:

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

- **Sample Variance**:

$$\sigma^2 = \frac{1}{n-1}\sum_{i}^{n} (x_i - \mu)^2$$

9.2 Naive Bayes Problems

- **Conditional Independence Assumption**: not likely to hold in real life, but can be improved upon using semi-conditionally independent with Bayesian networks.
- **Data Sparsity**: training data needs smoothing to handle unseen combinations in test data.

10 Decision Trees

- **Premise**: find a tree that is compact and generalize to new examples (as there is a trivial solution for training examples).
- **Hypothesis Space**: there are c^{k^d} distinct trees, where c is classes and d is features with k values.
- Note: Need to use greedy search to minimize search space, but may not return optimal solution.
- **Entropy**: probability that an arbitrary tuple in $x \in X$ belongs to class $c_i \in C$:

$$H(Y) = \sum_{c_i} P(x \in c_i) * \log_2(P(x \in c_i))$$

$$H(Y|X_i) = \sum_{x_i \in X_i} P(x_i) * H(Y|x_i))$$

- Note: uniform distribution maximizes entropy and deterministic data has no entropy.
- **Information Gain**: entropy needed to classify X after splitting X by some feature X_i, where $H(Y)$ is the information before splitting X:

$$Gain(X_i) = H(Y) - H(Y|X_i)$$

- **Tree Splits**: split by the attribute that maximizes information gain and recursively apply on resulting subtrees.
- **Break Point**: we can stop partitioning the tree when there are no labels left or the partition represents a single class.
- **Split Info**: given v is the number of discrete buckets, attribute A, and X_j is the jth bucket:

$$SplitInfo(A) = -\sum_{j=1}^{v} \frac{|X_j|}{|X|} * \log_2(\frac{|X_j|}{|X|})$$

- **Gain Ratio**: we use gain ratio as the metric for continuous valued attributes for the decision tree splitting choice, to prevent unfair advantages from more discrete buckets, which hurts generalization:

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}$$

- **Edge Cases**: if a leaf has no examples in testing, choose according to parent. If there are no attributes left (or choosing among parent), use majority vote.
- **Discretizing Data**: we convert the continuous attribute into discrete buckets by sorting the data and then consider whether to split among split point candidates. Alternatively, we learn a simple classifier to bin features into discrete classes.
- Note: to prevent overfitting by splitting into small buckets, we add a normalization term on the number of splits, split info.
- **Decision Stump**: threshold which splits data into two or more bins.
- **Postpruning**: remove branches from complete tree and test on validation set for result difference to counter overfitting.
- **Regularization**: penalize complex trees with an added $\alpha * complexity$ term.
- **Insights**: interpretable, fast, easy to tune parameters, but high prediction variance.

10.1 Random Forests

10.1.1 Random Forest Algorithm

- Iterate through $b \in Batches$.
- For each b, grow a forest tree T_b using a bootstrap sample of data size n and feature sample size m.
- For predictions, use desired ensemble scheme to output a final decision.
- **Bootstrap Sample**: random sample with replacement.

10.1.2 Insights

- Increases variability from random sampling of data and features.
- Slower runtime, but increased prediction variance and generally higher performance.
- Loses interpretability of decisions.
- Easy to tune parameters.

11 Support Vector Machines / Maximum Margin Classifier

- **Scalar Projection**:

$$proj_b(a) = \frac{a^T b}{||b||} = ||a||cos(a, b)$$

- **Hyperplane**: linear separator.

$$w^T x + b = 0$$

- **Hard Margin SVM**: satisfies the constraint below. Data must be linearly separable without error tolerance.

$$\min_i(y_i(w^T x_i + b)) = 1$$

- **Margin**: distance from hyperplane to support vector.

$$\gamma(h) = \frac{1}{||w||}$$

- **Support Vector**: input data points that satisfy the equality below. These points lie on the margin and uniquely determine the hyperplane.

$$y_i(w^T x_i + b) = 1$$

- **Hard Margin SVM Objective Function**: subjected to $\min_i(y_i(w^T x_i + b)) = 1$.

$$\min_{b,w}(\frac{1}{2} w^T w)$$

- **Quadratic Solver**: solves the following subject to $Au \geq c$:

$$\min_{u \in \mathbb{R}^q}(\frac{1}{2} u^T Q u + p^T u)$$

- For Hard Margin SVM (primal, linear case), use

$$u = \begin{bmatrix} b \\ w \end{bmatrix}, Q = \begin{bmatrix} 0 & 0_d^T \\ 0_d & I_d \end{bmatrix}, p = 0_{d+1}, A = \begin{bmatrix} y_1 & y_1 x_1^T \\ ... & ... \\ y_n & y_n x_n^T \end{bmatrix}, c = 1_d$$

- Final hypothesis where QP returns $\begin{bmatrix} b^* & w^* \end{bmatrix}^T$:

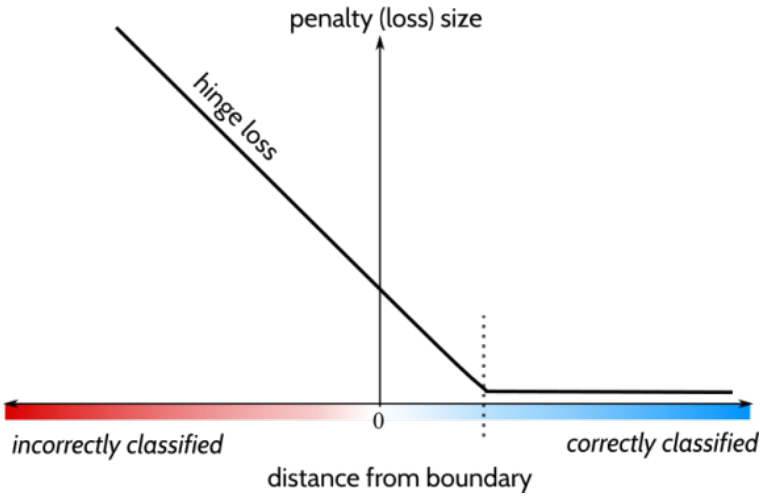
$$g(x) = sign(x^T w^* + b^*)$$

- **Nonlinear Transformation**: for SVM with nonlinear transformations, use $\Phi(x) = z$ in place of x.
- **Self-Regularization**: SVMs self-regularize as the objective minimizes the weight vector norm.
- **Primal Problem**: objective function is linear.
- **Dual Problem**: objective function is a linear combination of the variables in the primal problem.
- **Soft Margin SVM**: allows for margin violation $\epsilon_i \geq 0$ under the constraint below.

$$y_i(w^T x_i + b) \geq 1 - \epsilon_i$$

$$min_{b,w,\epsilon} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \epsilon_i$$

- Note: As $C \to 0$, we arrive at trivial solution $w \to 0$ since we allow for infinite error tolerance.
- Note: As $C \to \infty$, we arrive at the hard margin SVM solution since we allow for no error tolerance.
- **Hinge Loss**: $loss = 0$ when $x >= 1$. $loss$ is linearly increasing as $x \to -\infty$.



- **Error Tolerance**: can rewrite as Hinge Loss function:

$$E(b, w) = \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i(w^T x_i + b))$$

- **Soft Margin SVM Objective Function**:

$$\min_{b,w}(C * E(b, w) + \lambda w^T w)$$

12 Principal Component Analysis

- **Feature Reduction**: Use all features to create linear combinations as transformed features. Eliminates interpretability.
- **Supervised Setting Feature Reduction**: want to maximize the class discrimination.
- **Unsupervised Setting Feature Reduction**: want to minimize the information loss.
- **Feature Reduction Benefits**: visualization, data compression, noise removal.
- **Feature Selection**: select subset of original features. This is interpretable.
- **Principal Component Analysis**: reduces the dimensionality of a dataset by finding a new, smaller set of principal components, that retains most of the sample's variance. Used in unsupervised setting.
- **Geometric Picture of PCs**: PCs are a series of linear least squares fits to a sample, each orthogonal to all the previous. Orthogonal axes are uncorrelated, providing maximum information retention.
- **Covariance Matrix**: given $\bar{x}$ is the mean of data point x_i,

$$S = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})(x_i - \bar{x})^T$$

- **Algebraic Definition of PCs**: denote z_i as the ith PC. Given dataset $\{x_1, ..., x_n\} \in \mathbb{R}^p, x_j = (x_{1j}, ..., x_{pj})$,

$$z_{1j} = a_1^T x_j$$

- Constrained such that $a_1^T a_1 = 1$ and a_1 maximizes $Var(z_1)$.
- **Lagrange Transform**:

$$a_1^T S a_1 - \lambda(a_1^T a_1 - 1)$$

$$\frac{\delta L}{\delta a_1} = S a_1 - \lambda a_1 = 0$$

$$Var(z_1) = a_1^T S a_1 = \lambda_1$$

- For subsequent PCs, additional $Cov(z_2, z_1) = 0$ constraint and recursive definition:

$$Cov(z_2, z_1) = a_2^T \lambda a_1 = 0$$

- **Lagrange Transform**:

$$a_2^T S a_2 - \lambda(a_2^T a_2 - 1) - \gamma a_1^T a_2$$

$$\frac{\delta L}{\delta a_2} = Sa_2 - \lambda a_2 - \gamma a_1 = 0$$

$$Var(z_2) = a_2^T S a_2 = \lambda_2$$

- **kth Principal Component**:

$$Var(z_k) = a_k^T S a_k = \lambda_k$$

- **Reconstruction**: minimizes reconstruction loss for all linear projections size d using the following reconstruction matrix:

$$G(G^T X^T)$$

12.1 Challenges with High Dimensionality

- Machine learning and data mining techniques do not work well.
- **Curse of Dimensionality**: higher dimensional space has no neighbors.
- Model accuracy and efficiency degrade rapidly as the dimension increases. - Intrinsic dimension may be small.

12.2 Computing PCA

- Standardize $X \in \mathbb{R}^{n \times p}$ by subtracting $\bar{x}$.
- Calculate $S = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T = \frac{1}{n} X^T X$.
- Eigendecompose to find the d largest eigenvalues and eigenvectors.
- Construct transformation matrix $G = [a_1, ..., a_d]$.
- Apply PCA on $x \in \mathbb{R}^p \rightarrow G^T X \in \mathbb{R}^d$