COMS 4771 Exam 2 Cheat Sheet

Elizabeth Kushelevsky, Spring 2025

Linear Algebra.......... 2
Lp Norms.......... 4
Decision Trees.......... 5
Kernelization.......... 8
Support Vector Machines.......... 11

Linear Algebra

- Positive Definite: An n by n symmetric matrix A is PD if

  $$\forall \mathbf{x} \in \mathbb{R}^n \setminus \{0\} : \mathbf{x}^\top A \mathbf{x} > 0$$

  - Translation: A is PD if all of its eigenvalues are strictly positive.
  - A has full rank and A is invertible.
  - There exists an invertible PD matrix B of the same rank as A such that $B = A^T A$.

- Positive Semidefinite: An n by n matrix symmetric A is PSD if

  $$\forall \mathbf{x} \in \mathbb{R}^n : \mathbf{x}^\top A \mathbf{x} \geq 0$$

  - Translation: A is PSD if all of its eigenvalues are strictly nonnegative.
  - There exists a PSD matrix B of the same rank as A such that $B = A^T A$.

- Derivative of a linear function in vector form: for vectors a and w and scalar β,

  $$\frac{\partial}{\partial w} a^T w + \beta = a \qquad \text{(from Jacobian)}$$

- Derivative of a quadratic function in matrix form: for vector w and matrix A,

  $$\frac{\partial}{\partial w} w^T A w + \beta = \left(A^T + A\right) w \qquad \text{(from Jacobian)}$$

  In the case that A is symmetric, like all PSD matrices, this simplifies to $2Aw$.

- Second derivative of a linear function in matrix form: for vectors a and w and scalar β,

  $$\frac{\partial^2}{\partial w^2} a^T w + \beta = 0 \qquad \text{(from Hessian)}$$

- Second derivative of a quadratic function in matrix form: for vector w and matrix A,

  $$\frac{\partial}{\partial w} w^T A w + \beta = A^T + A \qquad \text{(from Hessian)}$$

  In the case that A is symmetric, like all PSD matrices, this simplifies to $2A$.

Let M be a symmetric positive semidefinite matrix.

1. Argue that for $\alpha > 0$, $M + \alpha I$ is positive definite. What are the eigenvalues w.r.t. the eigenvalues of M?
   New eigenvalues are $\lambda_i + \alpha$. If it is given that $\lambda_i \geq 0$ and $\alpha > 0$, then $\lambda_i + \alpha > 0$.

2. Find $\text{argmin}_w \left[\frac{1}{2} w^T (M + \alpha I) w - b \cdot w\right]$ for an arbitrary vector b.
   $\frac{d}{dw}\left(\frac{1}{2} w^T (M + \alpha I) w - b \cdot w\right) = (M + \alpha I) w - b$
   $(M + \alpha I) w - b = 0 \quad \rightarrow \quad (M + \alpha I) w = b$
   $(M + \alpha I)^{-1} (M + \alpha I) w = (M + \alpha I)^{-1} b$
   $w = (M + \alpha I)^{-1} b$

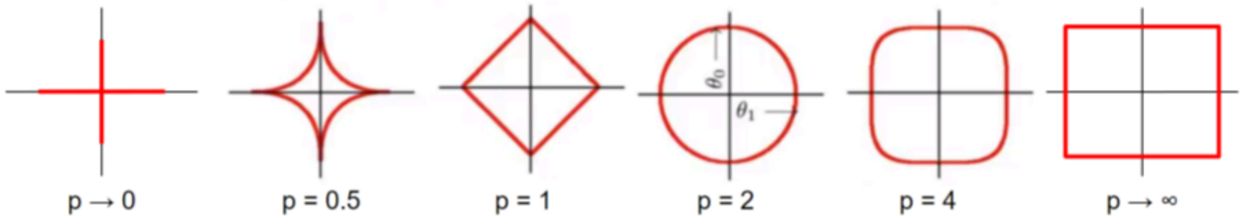
3. Find $\max_w \left[\frac{1}{2} w^T (M + \alpha I) w - b \cdot w\right]$ for an arbitrary vector b.
   Remember that this is a quadratic function shown as a PD matrix (convex set), which definitionally goes to ∞. By choosing any w orthogonal to b, the $b \cdot w$ term drops out and the $\frac{1}{2} w^T (M + \alpha I) w$ term goes to infinity.

L_p Norms

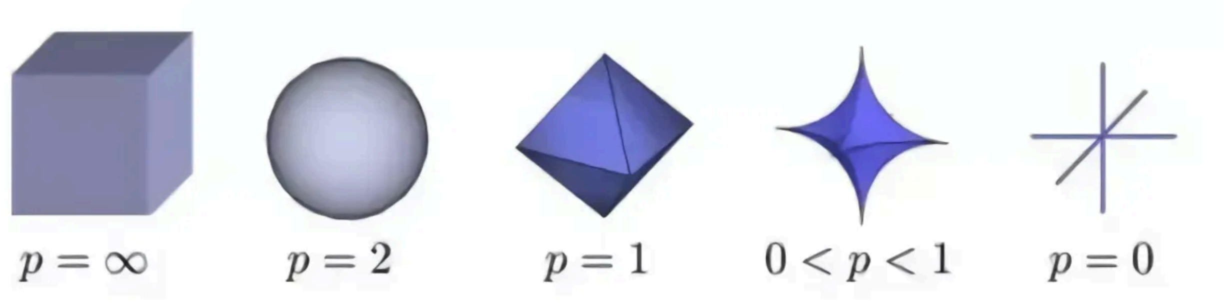
- Normed linear space L_p: a pair of objects (V, d) where V is the chosen vector space and d is a norm (length measurement). $d: V \times V = ||x|| \rightarrow \mathbb{R}$
- L_p norms have a specific form corresponding to their p: $||x||_p = \sqrt[p]{\sum_{i=0}^{n} \left|x_i\right|^p}$, where n is the dimensionality of V.
  - Example: $||x||_3 = (x_1^3 + x_2^3 + ... + x_n^3)^{1/3}$
- Properties of L_p norms:
  - $||x||_p \geq 0$
  - Only the zero vector in V has a norm of 0
  - $||\lambda x||_p = \lambda||x||_p \; \forall \, \lambda \in \mathbb{R}$ (closed under scalar multiplication)
  - $\left|\left|x_1 - x_2\right|\right|_p \leq \left|\left|x_1\right|\right|_p + \left|\left|x_2\right|\right|_p$ (triangle inequality)
    - $p < 0$ does not define a norm because it fails this property. This means that the set of values $||x||_p \leq 1$ is NOT a convex set.

- L_0 is a special norm: $||x||_0 = \left(\sum_{i=0}^{n} \left|x_i\right|^0\right)^{\infty}$ i.e. the number of nonzero components of x. Can be used to assess the vector's sparsity.
- L_∞ is a special norm: $||x||_\infty = max_i\left(\left|x_i\right|\right)$ i.e. the absolute value of the largest component of x.
- Common "unit spheres" of L_p norms (shown in $\mathbb{R}^2$):
  - L_1: we are looking for vectors $x = \left(x_1, x_2\right)$ s.t. $||x||_1 = \sqrt[1]{x_1 + x_2} = 1$. Starting from the elementary vectors, these unit vectors form a diamond.
  - L_2: same logic as before. Result is the unit circle.
  - L_n, $2 < n < \infty$: approaches the unit sphere of L_∞. A square with progressively pointier edges.
  - L_∞: must have at least one component of length 1. Result is a square.

In $\mathbb{R}^2$:



In $\mathbb{R}^3$:



Decision Trees

- Idea: split data along decision boundaries, then classify by region. To split smartly, pick the label that maximally reduces label uncertainty (helps accurate classification the most)

- Ways to measure uncertainty:
    - Classification error: where p_y is the fraction of training data labeled y, $u(C) = 1 - max_y\, p_y$
    - Entropy: corresponds to having 0 uncertainty when all info is known b/c of log.

      $$u(C) = \sum_{y \epsilon Y} p_y \, log \frac{1}{p_y}$$
    - Gini index:

      $$u(C) = \sum_{y \epsilon Y} p_y^2$$

Perceptron Algorithm

- Decision boundary is affine (form $w(x) = w \cdot x + w_0 = 0$)
- The linear classifier f originally has an offset: $sign(w \cdot x + x_0)$. To make the classifier pass through 0 and therefore closed under scalar multiplication, perform a "lifting" on it: $w' \cdot x' = (w, w_0) \cdot (x, 1)$
- Given training data $\{(x_1, y_1), \dots, (x_1, y_1)\}$, our goal is to find the weight vector w which minimizes training error:

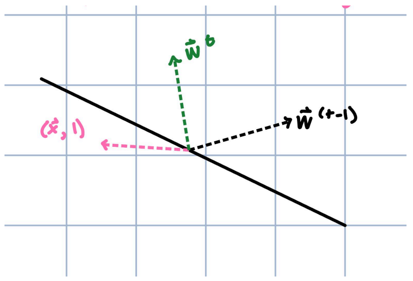
$$\text{argmin}_w \frac{1}{n} \sum_{i=1}^{n} 1\left[sign\left(w \cdot x_i\right) \neq y_i\right] \qquad \text{(Proportion of wrong classifications)}$$

$$= \text{argmin}_w \left(\sum_{x_i \, s.t. \, y_i = +1}^{n} 1\left[\left(x_i \cdot w\right) < 0\right] \right) + \left(\sum_{x_i \, s.t. \, y_i = -1}^{n} 1\left[\left(x_i \cdot w\right) \geq 0\right] \right)$$

- Algorithm: under the assumption that the data IS linearly separable,

```
Initialize $w^{(0)} = 0$
For $t$ = time index {
        Propose a $w$ and assess its training error.
        If $\exists\ (x, y)$ s.t. $sign(w^{(t-1)} \cdot x) \neq y$          (Misclassified point exists)
        $w^t = \begin{cases} w^{(t-1)} + x & \text{if } y = +1 \\ w^{(t-1)} - x & \text{if } y = -1 \end{cases}$
} Terminate when no mistakes remain.
```

- Geometrically, perceptron algorithm relies on the cosine inequality: $w \cdot x = ||w||||x|| \cos \theta$
    - Because cosine is positive for angles above the proposed decision boundary, it creates positive and negative sides of the boundary.



Here, point x is being misclassified as negative, so w^t will adjust to classify it as positive.

- Perceptron mistake bound: for data which is linearly separable by a unit-length boundary $w*$, there is an upper bound on the number of iterations the perceptron algorithm can make before it perfectly separates the data, $\left(\frac{R}{\gamma}\right)^2$, where R is the radius of the data (distance to farthest point) and γ is the margin (distance to closest point).

- R in the numerator: when the adjustment is by a large vector, we expect w to have a larger “swing”, meaning more misclassifications
- γ in the denominator: when the margin between oppositely-signed points is closer, it takes more iterations to find a w with no error

- Proof: we will analyze the closeness between an arbitrary current w^t and w^* using the dot product. Suppose this is not the first weight vector tried, and therefore $w^{(t-1)}$ classified a point inaccurately.

  $$w^t = \left(w^{(t-1)} + yx\right)$$

  Now, take the inner product with w^*.

  $$w^t \cdot w^* = \left(w^{(t-1)} + yx\right) \cdot w^*$$

  Because γ is the closest point to the decision boundary, offsetting $w^{(t-1)}$ to w^t means shifting the boundary by at least γ. This way, we get a lower bound for the RHS of the equation.

  $$\left(w^{(t-1)} + yx\right) \geq w^{(t-1)} + \gamma$$
  $$\left(w^{(t-1)} + yx\right) \cdot w^* \geq w^{(t-1)} \cdot w^* + \gamma$$
  $$w^t \cdot w^* \geq w^{(t-1)} \cdot w^* + \gamma$$

  Now we look at the norm of w^t.

  $$\left|\left|w^t\right|\right|^2 = \left|\left|w^{(t-1)} + yx\right|\right|^2$$
  $$= \left|\left|w^{(t-1)}\right|\right|^2 + 2y\left(w^{(t-1)} \cdot x\right) + ||yx||^2$$

  The second term is always non-positive since the prediction was wrong, so we take it at its maximum, 0, and it disappears from the equation. The upper bound for the distance from the sign-adjusted misclassified point is the radius R.

  $$\left|\left|w^t\right|\right|^2 \leq \left|\left|w^{(t-1)}\right|\right|^2 + R^2$$

  This inequality and the γ inequality hold true for all iterations of the algorithm.

  From the γ inequality, after T cycles of the algorithm, w must have shifted by at least $T * \gamma$.

  $$w^T \cdot w^* \geq w^0 \cdot w^* + T\gamma$$

  Because w^0 is defined to be the 0 vector, its term goes away.

$w^T \cdot w^* \geq T\gamma$

By the Cauchy-Schwarz inequality ($a \cdot b \leq ||a||||b||$),
$w^T \cdot w^* \leq ||w^T||||w^*||$ (w^* is a unit vector!)
$T\gamma \leq ||w^T||||w^*||$

From norm inequality, we have $||w^T||^2 \leq ||w^0|| + R^2T \rightarrow ||w^T||^2 \leq R^2T$.

$||w^T|| \leq R\sqrt{T}$ (which also bounds $T\gamma$)

$T \leq \left(\frac{R}{\gamma}\right)^2$

■

Given n linearly independent feature vectors in n dimensions, show that for any assignment to the binary labels, you can always construct a linear classifier with weight vector w which separates the points. Assume that the classifier has the form $sign(w \cdot x)$. Note that a square matrix composed of linearly independent rows is invertible.

To make the sign function able to classify binary data, simply assign $y = \pm 1$ as the binary outcomes and fill in the features and dimensions into a matrix X. We claim that a weight vector w exists s.t. Every point is correctly classified, i.e. $sign(X * w) = y$, which is true iff $(X * w) = y$. Then, since X is invertible, $w = X^{-1}y$ and w exists for any features and assignments.

Kernelization

- Introduced as an extension of the perceptron algorithm into non-linearly separable data. Theorem: all data is linearly separable in some space.
    - Shortcomings: data can be sent to an infinite-size vector where it can't be simply read, many points means a large computational cost, and the model cannot make a prediction on a posteriori data if it depends on individual a priori points

- Kernelization massages vectors so that their time-costly operations can be rewritten in terms of dot products, reducing computation time to $O(d)$. It *implicitly* considers vectors in higher dimensions through their similarity/dissimilarity without actually translating the vectors.
    - Method: rewrite your desired function using only dot products, then replace those dot products with a predetermined kernelized distance metric $k(x, z)$.

- Example: consider data that is quadratically separable in $\left(x_1, x_2\right)$. The linearization of points from $\left(x_1, x_2\right) \rightarrow \left(x_1^2, x_2^2\right)$ is a complicated operation:

  $$g(x) = w_1 x_1^2 + w_2 x_2^2 + w_3 x_1 x_2 + w_4 x_1 + w_5 x_2 + w_0$$
  $$= \sum_{i,j=1}^{d} \sum_{p+q\leq 2} w_{i,j}^{p,q} x_i^p x_j^q$$

  Instead, first define an explicit operation for $g(x)$ as the scaling of its coordinates:
  $$\phi(x) = \left(x_1^2, \ldots, x_d^2, \sqrt{2}x_1 x_2, \ldots, \sqrt{2}x_{d-1}x_d, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, 1\right)$$

  Then, the distance between two points $\phi(p)$, $\phi(q)$ becomes computable as $\phi(p) \cdot \phi(q) = (p \cdot q)^d = K(p, q)$.

- Kernelizing the perceptron algorithm: here, the costly operation we hope to make computable in linear time is the calculation of w^t from $w^{t-1} \ldots w^0$.

  $w = \sum_{k=1}^{n} \alpha_k y_k x_k$ (α $=$ number of mistakes on x_k)

  Instead, look at how data is classified based on this weighting:
  $$f(x) = sign(w \cdot x)$$
  $$= sign\left(\left(\sum_{k=1}^{n} \alpha_k y_k x_k\right) \cdot x\right) = sign\left(\sum_{k=1}^{n} \alpha_k y_k \left(x_k \cdot x\right)\right)$$
  In any non-linearly separable space which we hope to linearize, instead of carrying out the transformation $\phi(x)$, we can use the kernelized dot product.

  $f(x) = sign\left(\sum_{k=1}^{n} \alpha_k y_k \left(k(x_k, x)\right)\right)$ (For the user's choice of k)

- Some more common kernels:

Name	**Kernel Function (implicit dot product)**	**Feature Space (explicit dot product)**
Linear	$K(\mathbf{x},\mathbf{z}) = \mathbf{x}^T\mathbf{z}$	Same as original input space
Polynomial (v1)	$K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^T\mathbf{z})^d$	All polynomials **of** degree d
Polynomial (v2)	$K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^T\mathbf{z}+1)^d$	All polynomials **up to** degree d
Gaussian	$K(\mathbf{x},\mathbf{z}) = \exp(-\frac{\|\|\mathbf{x}-\mathbf{z}\|\|_2^2}{2\sigma^2})$	Infinite dimensional space
Hyperbolic Tangent (Sigmoid) Kernel	$K(\mathbf{x},\mathbf{z}) = \tanh(\alpha\mathbf{x}^T\mathbf{z}+c)$	(With SVM, this is equivalent to a 2-layer neural network)

Support Vector Machines

- General idea: find the *optimal* decision boundary as the midpoint of the two maximally wide boundaries. Minimize the inverse of this distance, $\frac{1}{2}||w^2||$, subject to the constraint for minimum clearance, $y_i(w_i \cdot x_i - b) \geq 1$.
    - If we wish to allow slack, the problem becomes min $\frac{1}{2}||w^2|| + C\sum_{i=1}^{n} \xi_i$ subject to $y_i(w_i \cdot x_i - b) \geq 1 - \xi_i$, where ξ_i is the slack allowed on point i and C is a user-controlled hyperparameter accounting for the tradeoff between slack and mistakes. Large $C \rightarrow$ assign large penalty to mistake
- SVM is a convex optimization problem, where a convex objective function f is minimized over a convex feasible region subject to constraints $g_i(x) \leq 0$.
    - It can be transformed into the Lagrangian function $L(x, \lambda)$, where λ is a scaling vector on the constraints.
      $L(x,\lambda) = f(x) + \sum_{i=1}^{n} \lambda_i g_i(x)$
    - Given the primal $p* = \min_x \max_\lambda L(x, \lambda)$ and the dual $d* = \max_\lambda \min_x L(x, \lambda)$, If the constraints can be strict (Slater's condition: $g_i(x) < 0 \ \forall g$ OR $g_i(x) \leq 0$ where g is affine $\rightarrow g$ creates a convex feasible region), $p* = d*$ (strong Lagrangian duality)
- For SVM, we have the following:
  Minimize w, b for $f(w) = \frac{1}{2}||w^2||$
  With constraints $y_i(w_i \cdot x_i - b) \geq 1$
  The Lagrangian is:
  $L(w, b, \alpha) = \frac{1}{2}||w^2|| + \sum_{i=1}^{n} \alpha_i(y_i(w_i \cdot x_i - b))$
  and $\min_{w,b}$ can be found by taking the gradient of L.
  $\frac{\partial}{\partial w}L(w, b, \alpha) = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0$
  $w = \sum_{i=1}^{n} \alpha_i y_i x_i$
  $||w^2|| = w^T w = \left(\sum_{i=1}^{n} \alpha_i y_i x_i\right)^T \left(\sum_{j=1}^{n} \alpha_j y_j x_j\right)$

$$= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j \left(x_i \cdot x_j\right)$$

Now we have the SVM standard form.

Maximize α_i on $\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j \left(x_i \cdot x_j\right)$

such that $\sum_{i=1}^{n} \alpha_i y_i, \alpha_i \geq 0$

- Because this maximization only performs dot products on x, it is kernelized.