Chapter 5: Support Vector Machines

- Linear SVM Classification
  - Intro: large margin classification, support vectors
  - Street analogy
  - Soft Margin Classification
    - Hard margin: requirement that only on one side of the line
      - Data must be linearly separable
      - Sensitive to outliers
    - Resolve by keeping street as wide as possible
    - Limit margin violations:
  - Sklearn SVM: C hyperparameter: C↓margin↑
  - Want to limit margin violations, but sometimes more violations allow model to be more generalizable
  - Sklearn models to use
    - LinearSVC(C=1, loss='hinge')
      - Regularizes bias term. So center training set first. Automatic with StandardScaler
      - Set loss='hinge' since not default
      - Set dual=False, unless features > len(label)
    - SVC(kernel='linear', C=1)
    - SGDClassifier(loss='hinge', alpha=1/(m*C)) ← Good for online classification tasks or huge datasets that don't fit into memory. Does not converge as fast as LinearSVC
- Nonlinear SVM Classification
  - Background:
    - Many datasets not linearly separable
    - Adding features (e.g., polynomial) can make separable
    - Add PolynomialFeatures in pipeline
  - Polynomial Kernel
    - Adding polynomial features works well with many models
    - Drawback: low degree may not handle complex data sets; high degree makes model slow → too many features
    - Kernel trick: get same result as having added many polynomial features without adding them
  - Adding Similarity Features
    - Another technique to deal with nonlinearity
    - Similarity function measures how much each instance resembles landmark
    - Gaussian RBF = exp(γ||x -l||^2)
    - Create landmark for every instance of dataset: multi-dimensional, but can
  - Gaussian RBF Kernel
    - Similarity features useful with any ML algo, but computationally expensive
    - Kernel trick
    - SVC: gamma ↑ narrows bell curve → irregular decision boundary

        - Underfitting increase gamma or C, Overfitting decrease
    - Which kernel?
        - Start with linear, then try rbf, then others using cross-validation & grid search
  - Computational Complexity
    - Kernel trick support: No for LinearSVC, Yes for SVC
    - Slow. Good for complex small to medium training sets
- SVM Regression
  - SVM supports linear and nonlinear classification and regression
  - Classification: fit largest street between two classes while limiting margin violations
  - Regression: fit as many instances on narrowest street while limiting margin violations
  - Hyperparameter ε controls margin width; adding more instances within margin doesn't affect predictions
  - LinearSVR for linear tasks, kernelized SVR for nonlinear
    - Large C little regularization, small C more
  - LinearSVC or LinearSVR vs. SVC or SVR
- Under the Hood
  - Background: how SVMs make predictions
  - Decision Function and Predictions
    - SVM classifier applies w to new x and if +ve, y is +ve
    - Linear SVC finds values of w and b that make margin as wide as possible while avoiding margin violations
  - Training Objective
    - Lower w, increase margin
    - Slack variable ζ: how much each instance allowed to violate margin
    - Conflict: make w small to increase margin, and slack variable small to reduce margin violations
    - C hyperparameter defines tradeoff between conflict
    - Minimize $\frac{1}{2} w^2 + C^* \Sigma \zeta$
      - Subject to $t(w^t x + b) >= 1 - \zeta$ and $\zeta >= 0$
  - Quadratic Programming (QP)
    - General programming problem
    - Minimize p: $\frac{1}{2} p^T H p + f^T p$
    - Subject to Ap <= b
      - P is n_p (# of parameters) dimensional vector
      - H is n_p x n_p matrix
      - f is n_p dimensional vector
      - A is n_c (# of constraints) x n_p matrix
      - b is n_c dimensional vector
  - The Dual Problem
    - Dual problem is closely related to constrained optimization problem.
    - Solution gives lower bound than primal
    - Both the same solution, dual is faster when training instances < # features

        - Kernel trick is possible with Dual
    - Kernelized SVM
        - Kernel trick: the dot product of transformed vectors is equal to the transformation of the dot product of the vectors given certain conditions
            - Insight: don't need to transform the vectors first, making it more computationally efficient
        - Kernel: function of capable of computing dot product of original vectors without having to compute transformation
        - Mercer's Theorem:
            - Given: function K
            - Conditions: K is continuous, symmetric K(a,b) = K(b,a)
            - Then: there exists another function that maps a and b onto another space where you can use K to compute dot products
            - Don't need to know mapping function, only that it exists
        - Can make predictions without knowing weights by plugging formula for weights into decision function for new instance of x
            - Converting primal problem to dual problem and using kernel trick
    - Online SVMs
        - Online learning is learning incrementally
        - Gradient descent converges more slowly than QP solver
        - Linear SVM classifier cost function
            - Weights plus hinge loss
            - $1/2w^tw + C \Sigma \max(0, 1 - t(w^tx + b))$
            - Hinge loss max(0, 1 - t)
                - 0 → t >=1