**Chapter 5 - Support Vector Machines**

Support Vector Machines (SVMs) as a powerful and versatile Machine Learning model capable of handling **linear or nonlinear classification, regression, and even outlier detection**. SVMs are particularly well-suited for classifying complex but small- or medium-sized datasets.

**Key Concepts:**

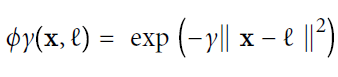
* **Fundamental Idea:** SVMs aim to fit the **widest possible street (margin)** between different classes. This is known as **large margin classification**.
* **Support Vectors:** The training instances located **on the edge of the street** are called support vectors. These instances are crucial as they fully determine the decision boundary; adding more training instances "off the street" does not affect the boundary.
* **Feature Scaling:** SVMs are **sensitive to the scales of the features**. It is important to scale the inputs before using SVMs to ensure the decision boundary looks appropriate and generalizes well.

**Linear SVM Classification:**

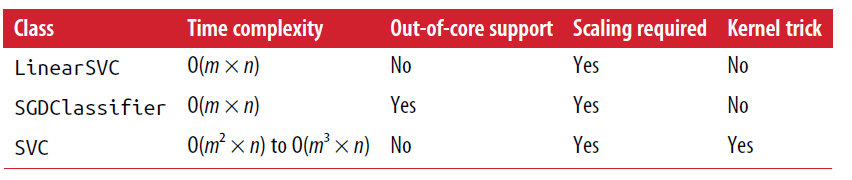
* Used when data is **linearly separable**.
* **Hard Margin Classification:** This approach strictly requires all instances to be *off* the street and on the correct side.
  + **Issues:** Only works if data is perfectly linearly separable and is very **sensitive to outliers**. An outlier can make it impossible to find a hard margin or drastically change the boundary.
* **Soft Margin Classification:** A more flexible approach that allows for some **margin violations** (instances in the middle of the street or on the wrong side).
  + Balances the goal of a large margin with minimizing margin violations.
  + Controlled by the **C hyperparameter**:
    - A smaller C leads to a **wider street** but allows more margin violations.
    - A higher C leads to a **smaller margin** but fewer margin violations.
  + If your SVM model is **overfitting**, you can regularize it by **reducing C**.
* **Implementations in Scikit-Learn:**
  + LinearSVC: Implements an optimized algorithm for linear SVMs. It is **much faster** than SVC(kernel="linear"), especially on large datasets. Does **not support the kernel trick** directly but is implicitly equivalent. Scales almost **linearly with the number of instances and features**. Requires centering the training set (automatic with StandardScaler). Set loss="hinge" and dual=False (unless n > m).
  + SVC(kernel="linear", C=1): Slower than LinearSVC for linear classification.
  + SGDClassifier(loss="hinge", penalty="l2"): Trains a linear SVM classifier using Stochastic Gradient Descent. Does not converge as fast as LinearSVC but can handle **huge datasets** that don't fit in memory (**out-of-core training**) or **online classification tasks**.

**Nonlinear SVM Classification:**

* Used when datasets are **not linearly separable**.
* **Techniques to handle nonlinearity:**
  + Adding **Polynomial Features**: Add powers of existing features (e.g., $x\_1^2$). Can make non-linear data linearly separable in a higher-dimensional space. Use PolynomialFeatures + StandardScaler + LinearSVC. The degree hyperparameter controls the polynomial degree, and coef0 controls the influence of high vs. low degree polynomials. Overfitting can be reduced by decreasing the polynomial degree.
  + Adding **Similarity Features**: Compute new features measuring how much each instance resembles specific "landmarks" using a similarity function (e.g., Gaussian RBF). The simplest approach is setting a landmark at every instance, but this creates many features, potentially too many for large datasets.
* **The Kernel Trick:** A mathematical technique that allows you to achieve a similar result as adding many nonlinear features, but without actually adding them. It computes the dot product of the implicitly transformed vectors using a **kernel function**.
  + Implemented in the SVC class.
  + SVC(kernel="poly", degree=3, coef0=1, C=1) for Polynomial features  saves us from explosion of features without adding them. The hyperparameter coef0 controls how much the model is influenced by high degree polynomials versus low-degree polynomials. Degree is the degree of polynomial and C is our regularization parameter. If your model is overfitting, reduce the polynomial degree. Conversely, if it is underfitting, increase the polynomial degree.
  + **Gaussian RBF Kernel:** A popular choice, SVC(kernel="rbf"). Controlled by hyperparameters **gamma (γ)** and **C**.



* + - Increasing gamma makes the decision boundary more irregular (higher variance, lower bias - **overfitting**), as each instance's influence range shrinks.
    - Decreasing gamma makes the boundary smoother (higher bias, lower variance - **underfitting**).
    - Gamma acts like a regularization hyperparameter. C also acts as a regularization parameter (decrease C for more regularization).
  + Other kernels exist (e.g., polynomial, sigmoid).
  + **Kernel Selection Rule:** Start with the **linear kernel** (LinearSVC), especially for large datasets. If the training set is not too large, try the **Gaussian RBF kernel** as it often works well. Experiment with others if computational power allows.
* **Computational Complexity of SVC (with Kernel Trick):** Typically between O(m² × n) and O(m³ × n). Becomes **very slow** for large numbers of training instances (hundreds of thousands). Scales well with the number of features, especially sparse features.

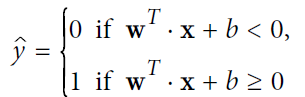


**SVM Regression:**

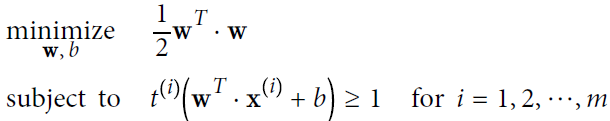
* Adapts the SVM approach for regression tasks.
* Objective: Fit the model such that as many instances as possible lie **on the street**, while minimizing violations (instances outside the street).
* The width of the street is controlled by the **epsilon (ϵ) hyperparameter**.
* The model is **ϵ-insensitive** because adding training instances within the margin does not affect predictions.
* Implemented in Scikit-Learn using:
  + LinearSVR for linear SVM Regression. Scales linearly with training set size.
  + SVR for nonlinear SVM Regression (supports the kernel trick). Gets slow for large training sets.
* SVMs can also be used for **outlier detection**.

**Under the Hood (Details):**

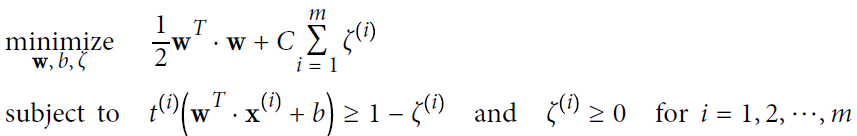
* SVMs often use a different notation for parameters: b for bias and w for the weight vector.
* The **decision function** for a linear SVM is wTx + b. A prediction is made based on the sign of this function. The **decision boundary** is where this function equals 0. i.e. The linear SVM classifier model predicts the class of a new instance **x** by simply computing the decision function  = **w***T* ・ **x** + *b* = *w*1 *x*1 + ⋯ + *wn xn* + *b*:



* Training a linear SVM classifier means finding the value of w and b that make this margin as wide as possible while avoiding margin violations (hard margin) or limiting them (soft margin). So we want to minimize . w . to get a large margin. However, if we also want to avoid any margin violation (hard margin), then we need the decision function to be greater than 1 for all positive training instances, and lower than –1 for negative training instances.
* **Hard margin linear SVM classifier objective**

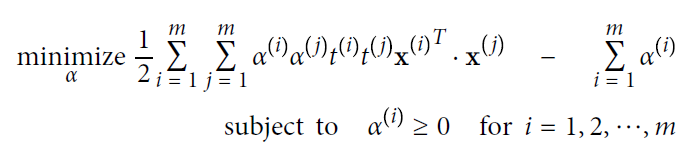


* **Soft margin linear SVM classifier objective**

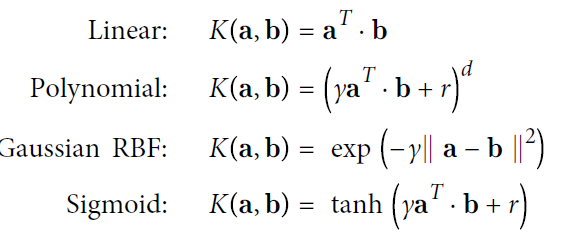


* The training problem is a **Quadratic Programming (QP)** problem.
* The **dual problem** is an alternative formulation that is faster to solve than the primal when the number of instances (m) is less than the number of features (n). It is also the formulation that makes the **kernel trick** possible.

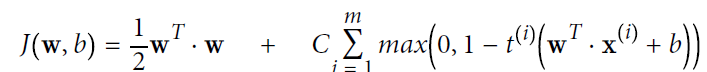
**Dual form of the linear SVM objective**



* The kernel trick replaces the computation of the dot product in a high-dimensional space ϕ(x(i))T ϕ(x(j)) with a kernel function K(x(i), x(j)) using only the original feature vectors.
* **Common kernels**



* When making predictions with a kernelized SVM, the decision function only involves computing the kernel function between the new instance and the **support vectors**.
* **Online SVMs** are available, including SGDClassifier for linear SVMs. Online kernelized SVMs are more complex.
* **Linear SVM classifier cost function**



* SVMs utilize the **hinge loss function**, max(0, 1 – t). It's not differentiable everywhere but can be used with Gradient Descent via subgradients.