Machine Learning I

Chapter 06 - Support Vector Machines (SVM)

Prof. Dr. Sandra Eisenreich

WS 2024/25

Hochschule Landshut

The task and when to use

- task: **binary hard classification**
- data: labeled, i.e. **supervised** learning. Instead of using 0 and 1 for the two classes, for SVM classification one usually uses -1 and 1, because some formula are easier this way. Note: Scaling data is important vor SVMs!
- model: depends... the basic model is **parametric.**
- complexity: slow for many training instances, see later.
- focus: **explainable**, flexibility depends on the variant (linear, Kernel, etc...)
- Note: Support Vector Machines are non-probabilistic models.

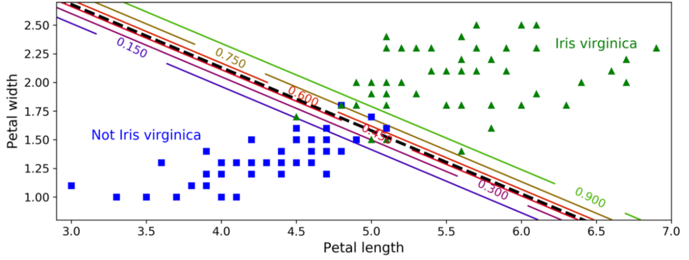
When to use:

- Linear SVM: only works if there is (more or less) a linear boundary between the classes in the training set (like with Logistic Regression).
- Kernel SVM: With nonlinear boundaries between the two classes.
- Don't use if you want class probabilities instead of hard labels. (or modify - look up Platt scaling)

Linear SVM Classification

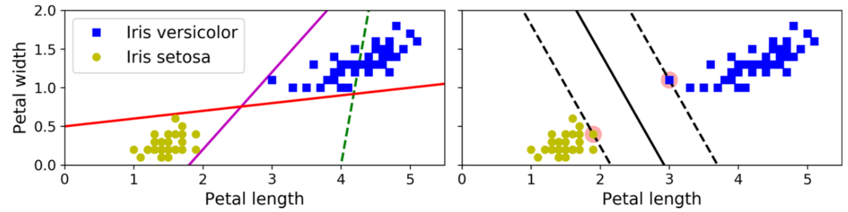
The idea: a decision boundary

A binary classification problem is called linearly separable, if the classes can be divided by a linear hyperplane in feature space.



Aim of SVM: Find not just any linear decision boundary, but the geometrically “best” one. What could that be?

Large margin classification



Answer: The line that is between classes and farthest away from all instances with some “safe space” = **margin** between the decision boundaries.

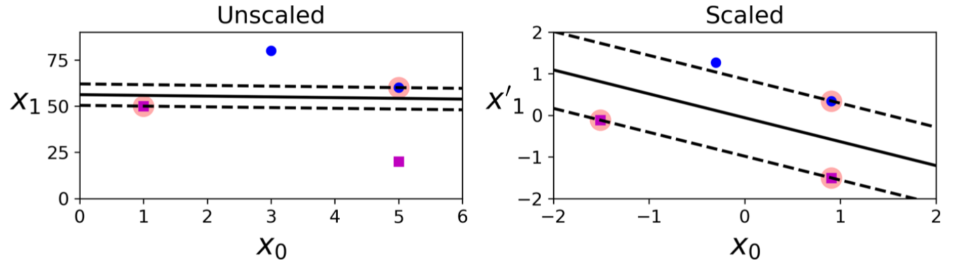
Solving a linearly separable classification problem by finding the decision boundary with the maximal margin from all datapoints is called large margin classification.

Support Vectors

- This geometric decision boundary is *fully determined* (or “supported”) by the instances that are closest to the decision boundary (pink in the image below).

> The instances that are closest to the decision boundary are called the support vectors. Sometimes the area between the support vectors is called the street.

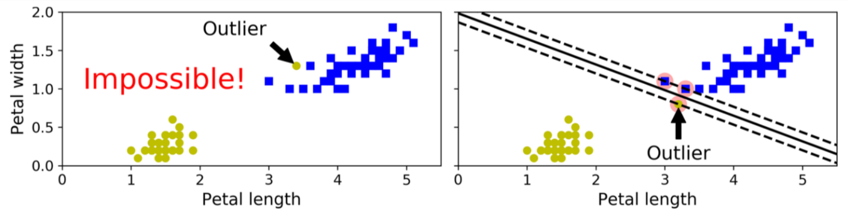
- The bigger the distance from the support vectors to the decision boundary, the better. Therefore, large margin classification is sensitive to scaling:



Hard Margin Classification

Hard margin classification: strictly impose that all instances of each class must be on their side of the decision boundary outside of the street.

Hard margin classification only works for linearly separable classification problems without outliers! One says it is “not robust to outliers“.

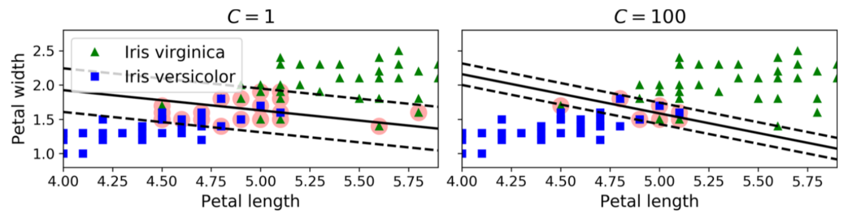


To avoid this, one usually uses a more flexible model: Soft Margin Classification.

Soft Margin Classification

- margin violation: an instance within the street (or on the wrong side of the decision boundary).
- soft margin classification: Allowing limited margin violations in a large margin classification problem

→ Trade off between number of margin violations and width of the street: There is a hyperparameter C which controls how much we penalize margin violations (large C → fewer violations, smaller margins):



Decision boundary and prediction function

We want a decision boundary like in Logistic Regression, i.e. a prediciton function

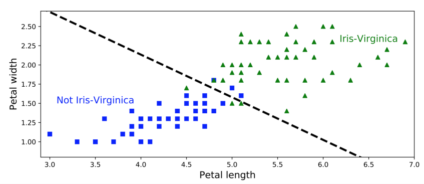
$$f(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \ldots + \theta_m x_m = b + \mathbf{w}^T \mathbf{x}$$

(if the input has m features) such that $f(\mathbf{x}) = 0$ is the decision boundary and

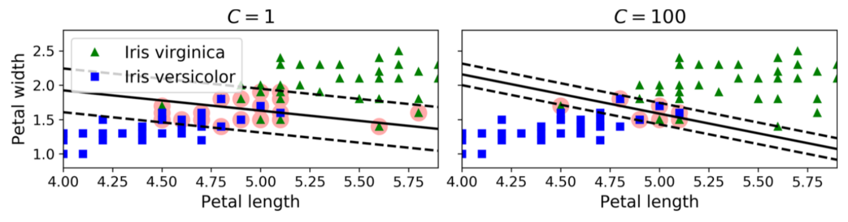
$$\hat{y} = \left\{ \begin{array}{cc} 1 & \text{if } f(\mathbf{x}) \geq 0 \\ -1 & \text{if } f(\mathbf{x}) < 0 \end{array} \right\} = \text{sign}(f)$$

Such a function $f = b + \mathbf{w}^T \mathbf{x}$ is called decision function.

But we want to find it in a different way than for logistic regression: not by optimizing cross-entropy, but geometrically.



Training Objectives



- **Objective 1**: all instances (or as many as possible for soft margrin classification) from the training set are on the correct side of the decision boundary.
- **Objective 2**: make the margin as big possible.

Note: This way, the predictions of SVM only depend on the support vectors! (the other instances don't change the decision boundary).

Objective 1: Training Instances on the correct side

If $\mathbf{x}$ has label y (either $+1$ or -1), we want correct predictions, i.e. ?

Objective 1:

- Hard margin SVM: f has to satisfy ? for all instances.
- Soft margin SVM: f should satisfy the above for as many instances as possible.

Objective 2: Maximal distance to the boundary

One can use geometry to show (see appendix of this chapter for calculations):

Let $f = b + \mathbf{w}^T\mathbf{x}$ be the decision function.

- The distance of a support vector $\mathbf{x}$ with prediction $\widehat{\mathbf{y}}$ from the decision boundary given by $f = 0$ is:

$$r = \frac{f(\mathbf{x})}{\widehat{\mathbf{y}}\|\mathbf{w}\|}.$$

- **Objective 2:** maximize $_{\mathbf{w}} \frac{f(\mathbf{x})}{\widehat{\mathbf{y}}\|\mathbf{w}\|}$ for all support vectors $\mathbf{x}$ on the margin line.

Question: What could be a practical problem with Objective 2?

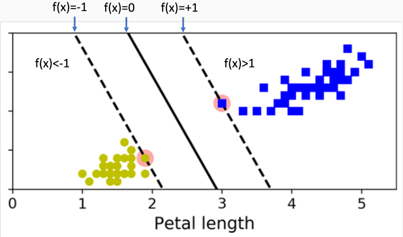
Answer: ?

Objective 2 revisited - rescaling f

Note: we only care if f is $\geq$ or $<$ than zero, not how large the value of f is.

$\Rightarrow$ scale the values of f (by multiplying all parameters with a constant) in a convenient way to make the **x**-part go away:

The decision function is chosen such that $f(\mathbf{x}) = \widehat{\mathbf{y}} \in \{+1, -1\}$ for all support vectors.



Since the support vectors are closest to the decision boundary and $|f|$ is increasing the further away you are from the decision boundary, we have ?? for all non-margin violations $\mathbf{x}$ $\Rightarrow$ Objective 1 ($f(\mathbf{x}) \cdot y > 0$) can be rewritten as ?

Objective 2 revisited - final version

maximizing $r = \frac{f(\mathbf{x})}{\widehat{\mathbf{y}}\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} \Leftrightarrow ???$

Objective 2, Final Version: minimize $\frac{1}{2}\|\mathbf{w}\|^2 = \frac{1}{2}\mathbf{w}^T\mathbf{w}$

Combining everything:

SVM classification objectives: minimize $\frac{1}{2}\|\mathbf{w}\|^2$ under the constraint $f(\mathbf{x}) \cdot y \geq 1$

- **hard margin SVM**: for all instances **x** with label y
- **soft margin SVM**: for most instances **x** with label y. Exceptions: margin violations

Hard-margin Support Vector Machine

A hard-margin Linear Support Vector Machine (SVM) is the ML model defined by:

- prediction function: $\widehat{\mathbf{y}} = \text{sign} f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$
- objective function: if the data are $\{(\mathbf{x}^{(i)}, y^{(i)}) | i = 1, \ldots, m\}$, then the objective is: minimize $_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w}$ subject to the constraint

$$y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \geq 1 \quad \text{for } i = 1, 2, \ldots, m$$

Attention: only works if the data is linearly separable, which usually isn't the case! So usually: soft-margin classification.

Motivation: Soft margin constraint

Note: For an instance to be on the right side of the margin means:

$$y^{(i)} \left(\mathbf{w}^T \mathbf{x}^{(i)} + b \right) \geq 1$$

Question: How can we tell an optimization problem that it should only allow "a few" margin violations? Hint: we could introduce one parameter $\zeta^{(i)} \in [0, \infty]$ per training instance $\mathbf{x}^{(i)}$ (=slack variables) which allows the instance to be on the wrong side of the margin

Answer: change the condition to ?

Soft-margin SVM

Soft-margin Linear SVM:

- prediction function: $\widehat{\mathbf{y}} = \operatorname{sign} f(\mathbf{x}) = \operatorname{sign}(\mathbf{w}^T\mathbf{x} + b)$
- objective function ($\zeta^{(i)} \geq 0$ are called the slack variables):

$$\text{minimize}_{\mathbf{w},b,\zeta^{(i)}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m}\zeta^{(i)} \text{ subject to the constraint:}$$

$$y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \geq 1 - \zeta^{(i)} \quad \text{and } \zeta^{(i)} \geq 0 \quad \text{for } i = 1, 2, \ldots, m,$$

- The hyperparameter C controls how much we penalize margin violations. Bigger C means less margin violations. $C = \infty$ is equivalent to hard-margin SVM.

This is a quadratic programming (QP) problem.

The dual problem

Optimierung: For every Quadratic Programming Problem, there is a Dual Problem (with Lagrange multipliers $\alpha^{(i)}$).

The “dual form“ of the hard-margin linear SVM objective is given by:

$$\text{minimize}_{\alpha} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} (\mathbf{x}^{(i)T} \mathbf{x}^{(j)}) - \sum_{i=1}^{m} \alpha^{(i)}$$

$$\text{subject to } \alpha^{(i)} \geq 0 \text{ for all } i = 1, \dots, m, \text{ and } \sum_{i=1}^{m} \alpha^{(i)} y^{(i)} = 0$$

Fact: Solving this is equivalent to solving the original SVM objective.

Note:

- there also is a dual form of the soft-margin SVM problem.
- the objective above depends on the data via the terms $\mathbf{x}^{(i)T}\mathbf{x}^{(j)}$! (Remember that!)

Complexity

We simply state this without a proof:

- Complexity of training a SVM classifier is between $O(m^2n)$ and $O(m^3n)$.
- If $v=$ number of support vectors, then inference time is $O(vn)$.

Note: Since SVMs can be of order three with respect to the number of instances, this becomes slow very quickly if you have lots of training examples.

SVM with Scikit-Learn

Don't forget to scale your data (e.g. with Scikit-Learns `StandardScaler`) before training a SVM!

Then you can use ScikitLearn's LinearSVC class to train the SVM:

```
from sklearn.svm import LinearSVC
svm_clf=LinearSVC(C=1, loss="hinge")
svm_clf.fit(X, y)
```

Alternatively, you could use `SVC(kernel="linear", C=1)`

Attention: SVMs are inherently hard classifiers, i.e. they only return the label, not a probability vector. However, one can "construct" probability vectors from SVM using Platt scaling, which is a neural network method. We will not cover this here. To get Scikit-Learn to use Platt scaling to output probability vectors, use `SVC(kernel="linear", C=1, probability=True)`.

Nonlinear SVM Classification

What to do with non-linearly separable data?

Many datasets are not linearly separable.

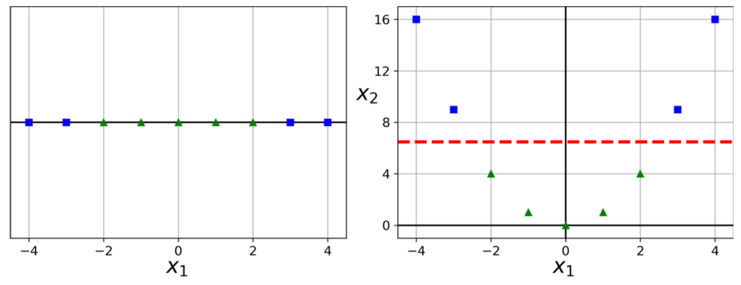
Possible solutions for nonlinear data:

- add more features, e.g. polynomial features, to make the feature space higher dimensional → could make data linearly separable. Easy to use, but can make model too slow because it quickly increases the number of features.
- use a non-linear function (called a kernel) “inside“ of the SVM Classification algorithm: “kernel trick“

Standard procedure: Try out linear SVM first, then if the training set is not too large try non-linear SVM with the kernel trick. If the result is not better, stick to linear SVMs.

Linear SVM with polynomial features

Just like in the case of Linear Regression we can add additional features to the dataset and hope that it will become linearly separable, like in this example:



To implement it in Scikit-Learn, we can use Scikit-Learn's PolynomialFeatures (plus StandardScaler) and a linearSVC.

Problem: Adding polynomial features makes the feature space dimensionality explode and the model slow!

Similarity Features

Recall that in the dual formulation of the SVM objective, we had the terms $\mathbf{x}^{(i)T}\mathbf{x}^{(j)}$. This is the scalar product of two vectors and can be seen as a linear similarity function of two vectors. Idea: replace these with non-linear similarity functions like the following:

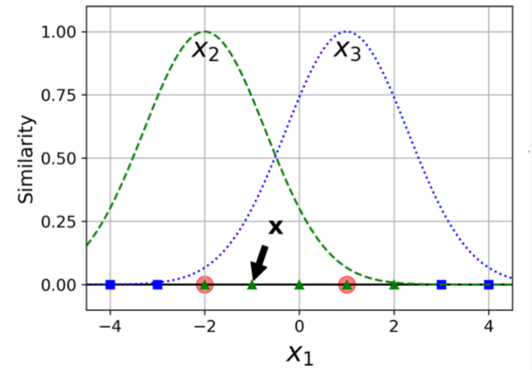
We define the Gaussian Radial Basis Function (also called Gaussian RBF or RBF-kernel) of two vectors as:

$$\phi_\gamma(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \quad = \quad \exp\left(-\gamma\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|_2^2\right) \text{ for a constant } \gamma \geq 0 \text{ (hyperparameter).}$$

Kernel SVM is defined like Linear SVM with the Dual Problem as an objective and the corresponding prediction functions, only with the terms $\mathbf{x}^{(i)T}\mathbf{x}^{(j)}$ replaced by $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ everywhere for a kernel K like the Gaussian RBF kernel. This method is called the "kernel trick".

Radial Basis Function

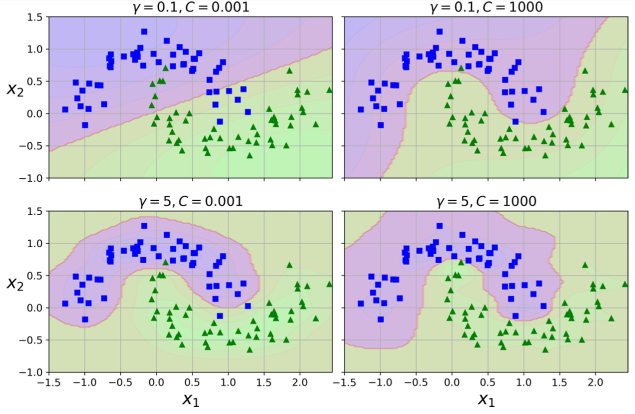
The Gaussian RBF-Kernel is a bell-shaped function which has value close to 1 if $\mathbf{x}^{(i)}$ is close to $\mathbf{x}^{(j)}$ and value close to 0 if they are far apart. The RBF kernels as functions in $\mathbf{x}^{(i)}$ for landmarks $\mathbf{x}^{(j)} = -2$ and $\mathbf{x}^{(j)} = 1$:



Other kernels exist but are used much more rarely.

The hyperparameter γ

The hyperparameter γ changes the shape of the bell: increasing γ makes the bell narrower and each instance's range of influence is smaller. Therefore, the decision boundary will end up more irregular, wiggling around individual instances



Other Kernels

Instead of the Gaussian RBF kernel, you can use other so-called Mercer kernels. Common Mercer kernels are:

- the linear kernel $K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T\mathbf{b}$ (this gives Linear SVM)
- the polynomial kernel of degree d with hyperparameters $\gamma \in \mathbb{R}, r \in \mathbb{R}$:

$$K(\mathbf{a}, \mathbf{b}) = (\gamma\mathbf{a}^T\mathbf{b} + r)^d$$

- the Sigmoid kernel with hyperparameters $\gamma \in \mathbb{R}, r \in \mathbb{R}$: $K(\mathbf{a}, \mathbf{b}) = \tanh\big(\gamma\mathbf{a}^T\mathbf{b} + r\big)$

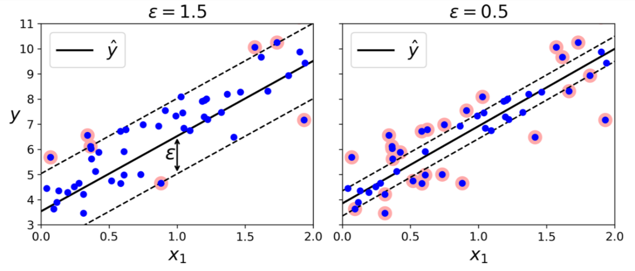
linear SVMs with Scikit-Learn

```
rbf_kernel_svm_clf = Pipeline([
                ("scaler", StandardScaler()),
                ("svm_clf", SVC(kernel="rbf", gamma=5, C=0.001))
        ])
rbf_kernel_svm_clf.fit(X, y)
```

SVM Regression

You can also use SVM for regression instead of classification: try to fit as many instances between the margin lines while limiting instances off the street. The width of the street is controlled by the hyperparameter ϵ.Bigger ϵ will make the margin wider.

Adding more training instances within the margin doesn't affect the model's prediction, so the model is said to be ϵ-insensitive.



Of course, you once again can use non-linear kernels.

```
from sklearn.svm import SVR
svm_poly_reg = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
svm_poly_reg.fit(X, y)
```

Chapter 06 Summary

Chapter 06 Summary - Hard-margin SVMs

- task/setting of linear SVMs is the same as for Logistic Regression: binary classification, supervised learning. Condition: there is a linear decision boundary between the two classes.
- fundamental idea: construct the geometrically best decision boundary, such that
  - all (hard margin SVM) or most (soft margin SVM) training instances are on the right side of the decision boundary, and
  - the street between the classes is as wide as possible. The boundary of the street is called margin, and the instances on the margin are called support vectors. The model depends on these instances.
- A hard-margin Linear Support Vector Machine (SVM) is the ML model defined by:
  - prediction function: $\widehat{\mathbf{y}} = h_\theta(\mathbf{x}) = \text{sign} f(\mathbf{s}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$
  - objective function for data $\{(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \ldots, m\}$:
    $\text{minimize}_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w},$ subject to the constraint $y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \geq 1 \quad \text{for } i = 1, 2, \ldots, m$

  It only works if the data is linearly separable!

Chapter 06 Summary - Soft-margin SVMs

- A soft-margin Linear Support Vector Machine (SVM) is the ML model defined by:
  - the same prediction function as hard-margin linear SVM
  - objective function is changed to

$$\text{minimize}_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m}\zeta^{(i)}, \text{subject to the constraint}$$

$$y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \geq 1 - \zeta^{(i)} \quad \text{and } \zeta^{(i)} \geq 0 \quad \text{for } i = 1, 2, \ldots, m,$$

  where $\zeta^{(i)} \geq 0$ are called the slack variables.

  The hyperparameter C controls how many margin violations are allowed - small C allows more violations.

- SVMs are trained using Quadratic Programming.
- The dual problem of the SVM objective can be used to optimize SVMs. The dual problem contains the terms $\mathbf{x}^{(i)T}\mathbf{x}^{(j)}$. Replacing these linear terms with non-linear Mercer kernels like the ones below yields non-linear/Kernel SVM.

Chapter 06 Summary - Kernel SVM and Regression

- Kernel SVM is used for binary classification with non-linear decision boundary.
- You can use e.g. the following Mercer kernels for Kernel SVMs:
  - the Gaussian Radial Basis Function (also called Gaussian RBF or RBF-kernel) of two vectors:

$$\phi_\gamma(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \quad = \quad \exp\left(-\gamma \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|_2^2\right) \text{ for a constant } \gamma \geq 0.$$

  - the linear kernel $K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T \mathbf{b}$ (this gives Linear SVM)
  - the polynomial kernel of degree d with hyperparameters $\gamma \in \mathbb{R}, r \in \mathbb{R}$:

$$K(\mathbf{a}, \mathbf{b}) = (\gamma \mathbf{a}^T \mathbf{b} + r)^d$$

  - the Sigmoid kernel with hyperparameters $\gamma \in \mathbb{R}, r \in \mathbb{R}$: $K(\mathbf{a}, \mathbf{b}) = \tanh\left(\gamma \mathbf{a}^T \mathbf{b} + r\right)$
- One can also use SVMs for Regression, by fitting as many instances as possible not outside of the street, but inside the street.
- Complexity of SVM Classifiers:
  - Complexity of training a SVM classifier is between $O(m^2 n)$ and $O(m^3 n)$.
  - If $v =$ number of support vectors, then inference time is $O(vn)$.