Applied Machine Learning

Support Vector Machines

Yasin Ceran

Motivation

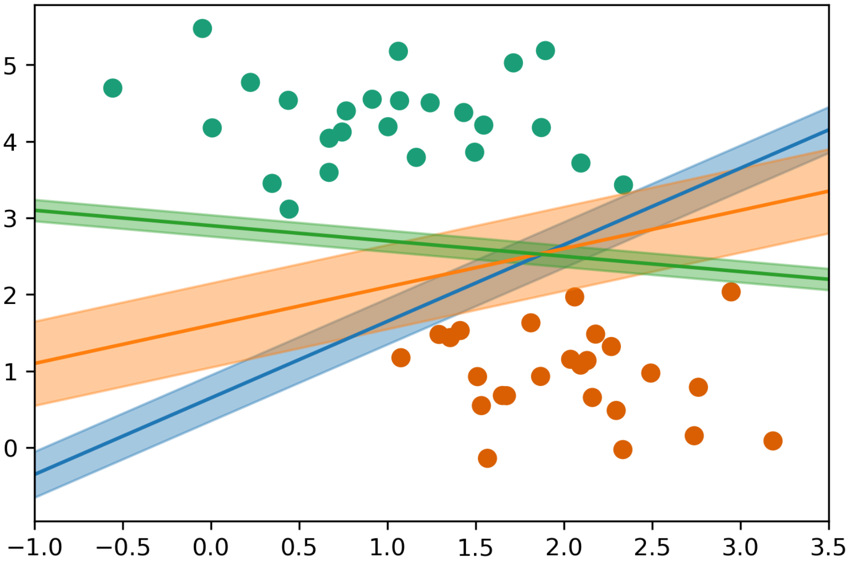
- Go from linear models to more powerful nonlinear ones.
- Keep convexity (ease of optimization).
- Generalize the concept of feature engineering.

Reminder on Linear SVM

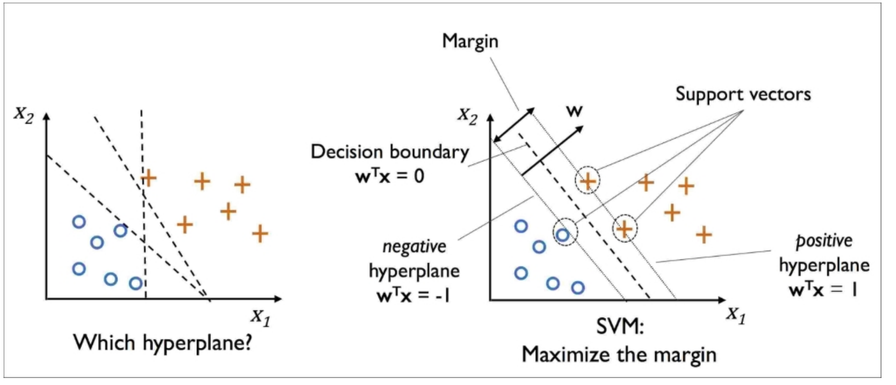
$$\min_{w \in \mathbb{R}^p} C \sum_{i=1}^{n} \max(0, 1 - y_i w^T \mathbf{x}) + ||w||_2^2$$

$$\hat{y} = \text{sign}(w^T \mathbf{x})$$

Max-Margin and Support Vectors



Maximum Margin Intuition (1)



-The margin is defined as the distance between the separating hyperplane (decision boundary) and the training samples that are closest to this hyperplane, which are the so-called support vectors.

Maximum Margin Intuition (1)

-To get an idea of the margin maximization, let's take a closer look at those positive and negative hyperplanes that are parallel to the decision boundary, which can be expressed as follows:

- $w_0 + \mathbf{w}^T \mathbf{x}_{pos} = +1$ (1) and
- $w_0 + \mathbf{w}^T \mathbf{x}_{neg} = -1$ (2)
- If we subtract (2) from (1) we have: $\mathbf{w}^T (\mathbf{x}_{pos} - \mathbf{x}_{neg}) = 2$
- We can normalize this equation by the length of the vector $\mathbf{w}$:

$$||\mathbf{w}|| = \sqrt{\sum_{j=1}^{m} w_j^2}$$

- So we arrive at the following equation:

$$\frac{\mathbf{w}^T (\mathbf{x}_{pos} - \mathbf{x}_{neg})}{||\mathbf{w}||} = \frac{2}{||\mathbf{w}||}$$

- The left side is the distance between the positive and negative hyperplanes, i.e., margin

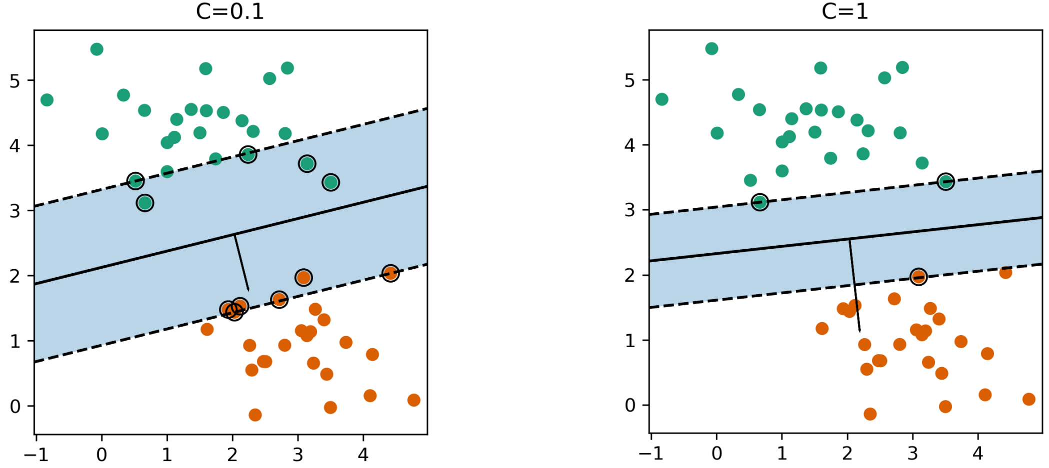
Max-Margin and Support Vectors

$$\min_{w \in \mathbb{R}^p} C \sum_{i=1}^{n} \max(0, 1 - y_i w^T \mathbf{x}) + ||w||_2^2$$

$$\text{Within margin} \Leftrightarrow y_i w^T x < 1$$

Smaller $w \Rightarrow$ larger margin

Max-Margin and Support Vectors



Reformulate Linear Models

- Optimization Theory

$$w = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$$

(alpha are dual coefficients. Non-zero for support vectors only)

$$\hat{y} = \mathrm{sign}(w^T \mathbf{x}) \Longrightarrow \hat{y} = \mathrm{sign}\left(\sum_{i}^{n} \alpha_i (\mathbf{x}_i^T \mathbf{x})\right)$$

$$\alpha_i <= C$$

Introducing Kernels

$$\hat{y} = \operatorname{sign}\left(\sum_{i}^{n} \alpha_i (\mathbf{x}_i^T \mathbf{x})\right) \longrightarrow \hat{y} = \operatorname{sign}\left(\sum_{i}^{n} \alpha_i (\phi(\mathbf{x}_i)^T \phi(\mathbf{x}))\right)$$

$$\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \longrightarrow k(\mathbf{x}_i, \mathbf{x}_j)$$

k positive definite, symmetric $\Rightarrow$ there exists a ϕ! (possilby ∞-dim)

Examples of Kernels

$$k_{\text{linear}}(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$

$$k_{\text{poly}}(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^d$$

$$k_{\text{rbf}}(\mathbf{x}, \mathbf{x}') = \exp(\gamma ||\mathbf{x} - \mathbf{x}'||^2)$$

$$k_{\text{sigmoid}}(\mathbf{x}, \mathbf{x}') = \tanh\big(\gamma \mathbf{x}^T \mathbf{x}' + r\big)$$

$$k_{\cap}(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{p} \min(x_i, x_i')$$

- If k and k' are kernels, so are $k + k'$, kk', ck', $\ldots$

Polynomial Kernel vs Features

$$k_{\text{poly}}(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^d$$

Primal vs Dual Optimization

Explicit polynomials → compute on `n_samples * n_features ** d`

Kernel trick → compute on kernel matrix of shape `n_samples * n_samples`

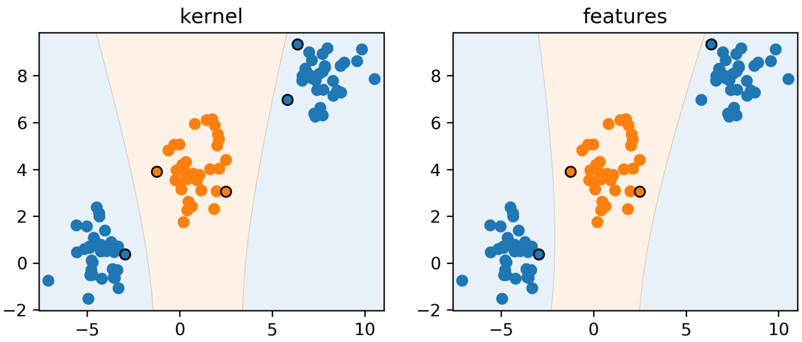
For a single feature:

$$(x^2, \sqrt{2}x, 1)^T (x'^2, \sqrt{2}x', 1) = x^2 x'^2 + 2xx' + 1 = (xx' + 1)^2$$

Poly kernels with sklearn

```
poly = PolynomialFeatures(include_bias=False)
X_poly = poly.fit_transform(X)
print(X.shape, X_poly.shape)
print(poly.get_feature_names())
```

```
((100, 2), (100, 5))
['x0', 'x1', 'x0^2', 'x0 x1', 'x1^2']
```



Understanding Dual Coefficiants

```
linear_svm.coef_
#array([[0.139, 0.06, -0.201, 0.048, 0.019]])
```

$$y = \text{sign}(0.139x_0 + 0.06x_1 - 0.201x_0^2 + 0.048x_0x_1 + 0.019x_1^2)$$

```
linear_svm.dual_coef_
#array([[-0.03, -0.003, 0.003, 0.03]])
linear_svm.support_
#array([1,26,42,62], dtype=int32)
```

$$y = \text{sign}(-0.03\phi(\mathbf{x}_0)^T\phi(x) - 0.003\phi(\mathbf{x}_{26})^T\phi(\mathbf{x}) + 0.003\phi(\mathbf{x}_{42})^T\phi(\mathbf{x}) + 0.03\phi(\mathbf{x}_{63})^T\phi(\mathbf{x}))$$

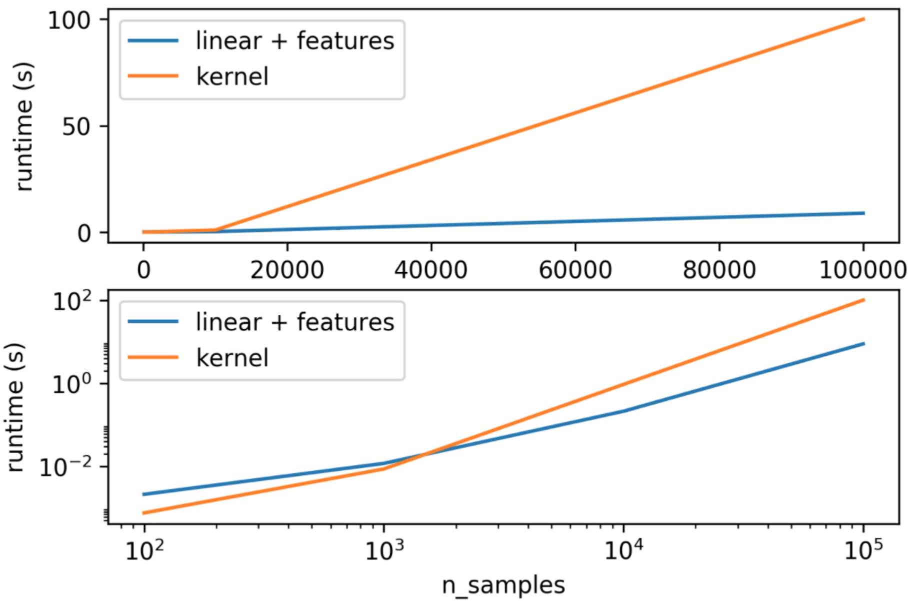
With Kernel

$$y = \operatorname{sign}\left(\sum_{i}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x})\right)$$

```
poly_svm.dual_coef_
# array([[-0.057, -0. , -0.012, 0.008, 0.062]])
poly_svm.support_
# array([1,26,41,42,62], dtype=int32)
```

$$y = \operatorname{sign}(-0.057(\mathbf{x}_1^T\mathbf{x} + 1)^2 - 0.012(\mathbf{x}_{41}^T\mathbf{x} + 1)^2$$
$$+ 0.008(\mathbf{x}_{42}^T\mathbf{x} + 1)^2 + 0.062 * (\mathbf{x}_{63}, \mathbf{x} + 1)^2$$

Runtime Considerations



Kernels in Practice

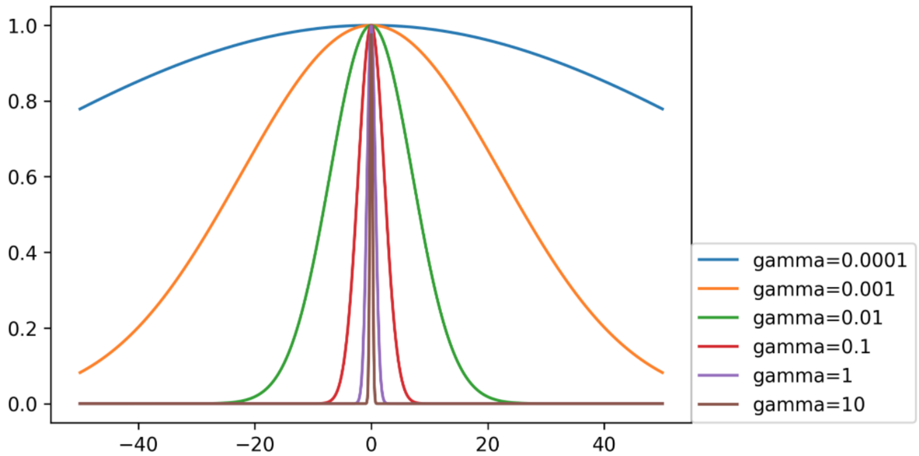
- Dual coefficients less interpretable
- Long runtime for “large” datasets (100k samples)
- Real power in infinite-dimensional spaces: rbf!
- Rbf is “universal kernel” - can learn (aka overfit) anything.

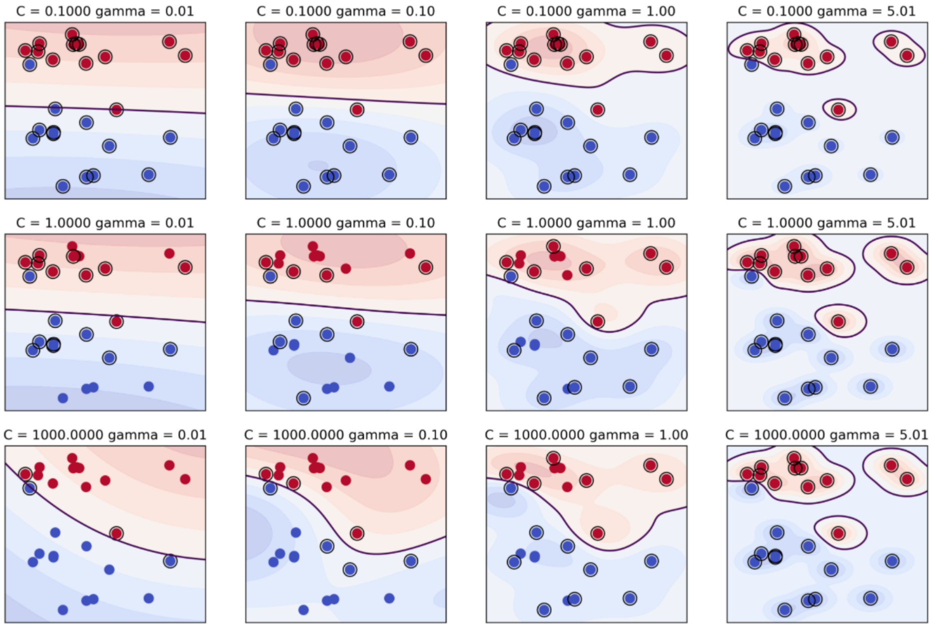
Preprocessing

- Kernel use inner products or distances.
- StandardScaler or MinMaxScaler ftw
- Gamma parameter in RBF directly relates to scaling of data – default only works with zero-mean, unit variance.

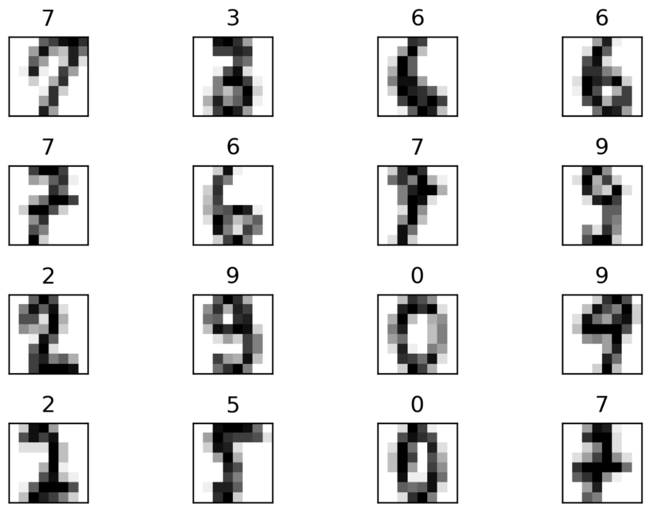
Parameters for RBF Kernels

- Regularization parameter C is limit on alphas (for any kernel)
- Gamma is bandwidth: $k_{\text{rbf}}(\mathbf{x}, \mathbf{x}') = \exp(\gamma||\mathbf{x} - \mathbf{x}'||^2)$




C = 0.1000 gamma = 0.01
C = 0.1000 gamma = 0.10
C = 0.1000 gamma = 1.00
C = 0.1000 gamma = 5.01
C = 1.0000 gamma = 0.01
C = 1.0000 gamma = 0.10
C = 1.0000 gamma = 1.00
C = 1.0000 gamma = 5.01
C = 1000.0000 gamma = 0.01
C = 1000.0000 gamma = 0.10
C = 1000.0000 gamma = 1.00
C = 1000.0000 gamma = 5.01

```python
from sklearn.datasets import load_digits
digits = load_digits()
```



Scaling and Default Params

```
gamma : float, optional (default = "auto")
  Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.
  If gamma is 'auto' then 1/n_features will be used
```

```
scaled_svc = make_pipeline(StandardScaler(), SVC())
print(np.mean(cross_val_score(SVC(), X_train, y_train, cv=10)))
print(np.mean(cross_val_score(scaled_svc, X_train, y_train, cv=10)))
```

```
0.578
0.978
```

```
gamma = (1. / (X_train.shape[1] * X_train.std()))
print(np.mean(cross_val_score(SVC(gamma=gamma), X_train, y_train, cv=10)))
```

```
0.987
```

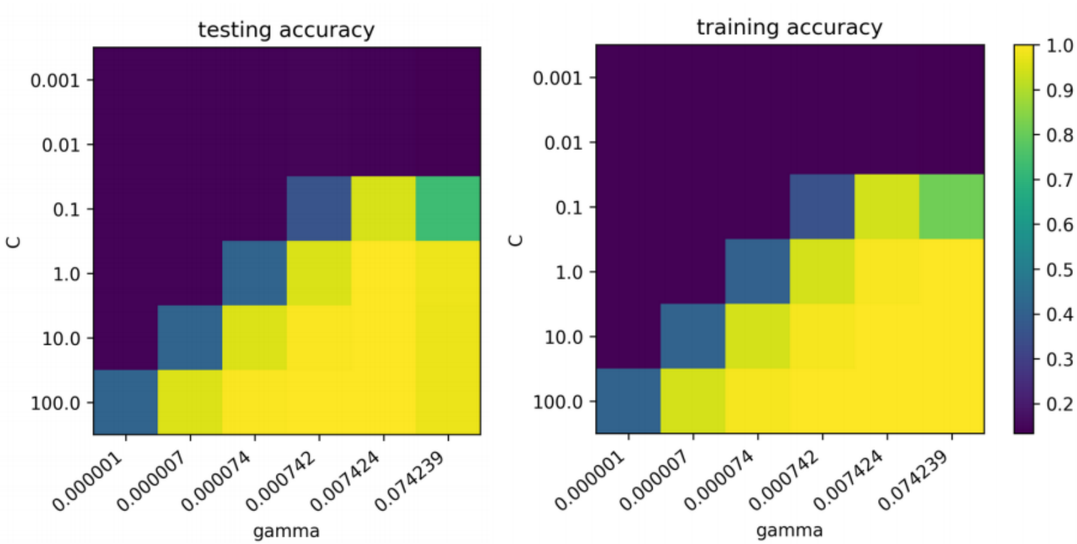
Grid-Searching Parameters

```
param_grid = {'svc__C': np.logspace(-3, 2, 6),
              'svc__gamma': np.logspace(-3, 2, 6) / X_train.shape[0]}
param_grid
```

{'svc__C': array([0.001, 0.01 , 0.1 , 1. , 10. , 100.]),
'svc__gamma': array([0.000001, 0.000007, 0.000074, 0.000742, 0.007424,
0.074239])}

```
grid = GridSearchCV(scaled_svc, param_grid=param_grid, cv=10)
grid.fit(X_train, y_train)
```

Grid-Searching Parameters



Questions ?