

SVMs

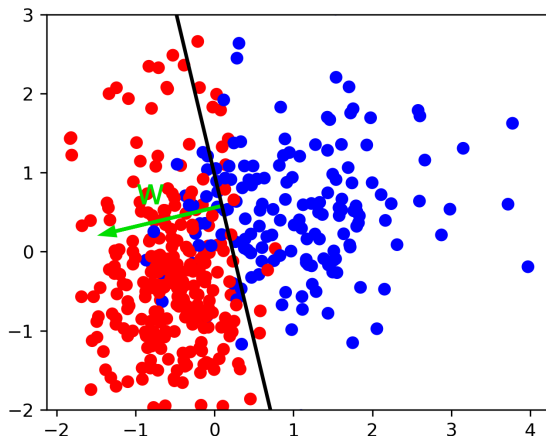
Presented by Yasin Ceran

Table of Contents

1 Linear Models for Binary Classification

2 Kernel SVMs

Linear Binary Classification

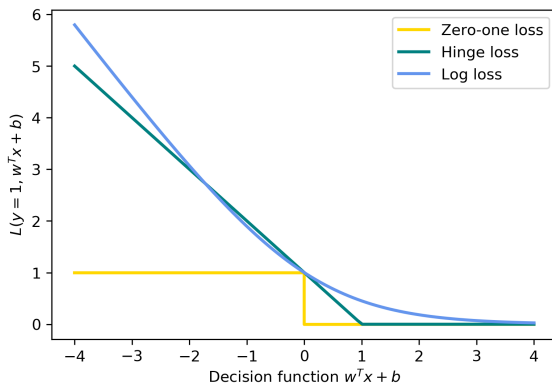


$$\hat{y} = \text{sign}(w^T \mathbf{x} + b) = \text{sign} \left(\sum_i w_i x_i + b \right)$$

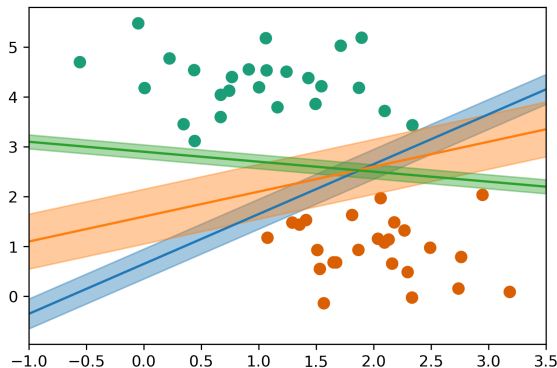
Picking A Loss

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

$$\min_{w \in \mathbb{R}^p, b \in \mathbb{R}} \sum_{i=1}^n 1_{y_i \neq \text{sign}(w^T \mathbf{x} + b)}$$

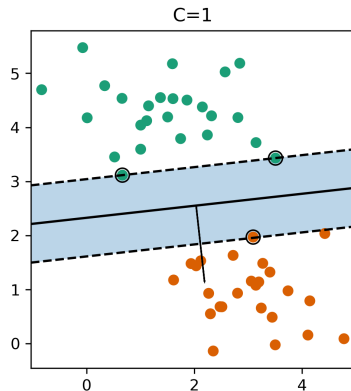
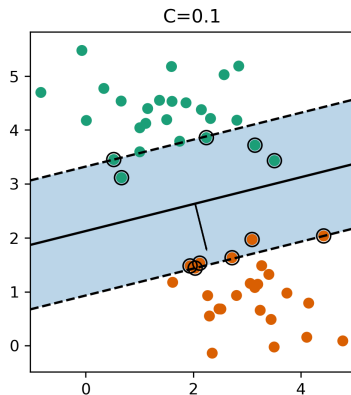


Max-Margin and Support Vectors (1)

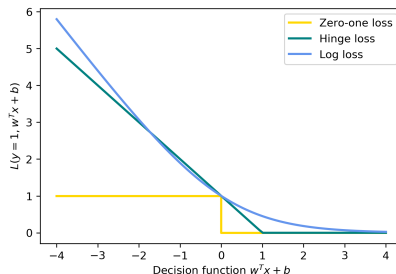


- $\min_{w \in \mathbb{R}^p, b \in \mathbb{R}} C \sum_{i=1}^n \max(0, 1 - y_i(w^T x + b)) + \|w\|_2^2$
- Within margin $\Leftrightarrow y_i(w^T x + b) < 1$
- Smaller $w \Rightarrow$ larger margin

Max-Margin and Support Vectors (2)



Logistic Regression vs SVM



- $\min_{w \in \mathcal{P}, b \in \mathbb{R}} C \sum_{i=1}^n \log(\exp(-y_i(w^T \mathbf{x}_i + b)) + 1) + \|w\|_2^2$
- $\min_{w \in \mathcal{P}, b \in \mathbb{R}} C \sum_{i=1}^n \max(0, 1 - y_i(w^T \mathbf{x}_i + b)) + \|w\|_2^2$
- Do you need probability estimates?
 - If yes, use Logistic Regression
 - If it doesn't matter, try either/both
- Need compact model or believe solution is sparse, use L_1 .

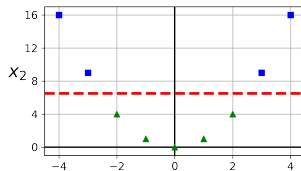
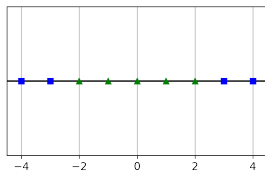
Table of Contents

1 Linear Models for Binary Classification

2 Kernel SVMs

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

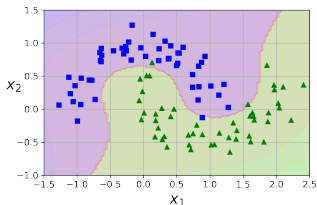
$$\min_{w \in \mathbb{R}^p, b \in \mathbb{R}} C \sum_{i=1}^n \max(0, 1 - y_i(w^T \mathbf{x} + b)) + \|w\|_2^2$$

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

```

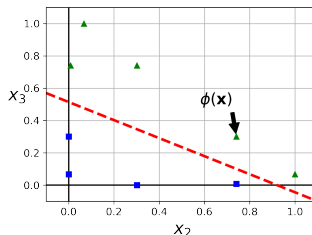
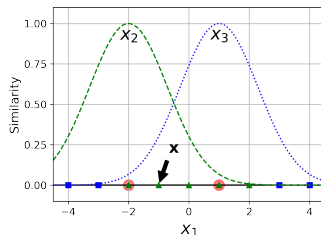
1  #####POLYNOMIAL FEATURES#####
2  from sklearn.datasets import
↳ make_moons
3  from sklearn.pipeline import
↳ Pipeline
4  from sklearn.preprocessing import
↳ PolynomialFeatures, StandardScaler
5  from sklearn.svm import LinearSVC
6
7  polynomial_svm_clf = Pipeline([
8      ("poly_features",
↳ PolynomialFeatures(degree=3)),
9      ("scaler",
↳ StandardScaler()),
10     ("svm_clf", LinearSVC(C=10,
↳ loss="hinge",
↳ random_state=42))
11 ])
12
13 polynomial_svm_clf.fit(X, y)

```



Similarity Features

- You can use a *similarity function* to measure how much each instance resembles a particular *landmark*
- Gaussian Radial Basis Function (RBF)*: $\phi_\gamma(\mathbf{x}, l) = \exp(\gamma \|\mathbf{x} - l\|^2)$



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)

$$\hat{y} = \text{sign}(w^T \mathbf{x}) \implies \hat{y} = \text{sign} \left(\sum_i^n \alpha_i (\mathbf{x}_i^T \mathbf{x}) \right)$$

$$\alpha_i \leq C$$

Introducing Kernels

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

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

Examples of Kernels

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

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

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

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

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

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

Polynomial Kernel vs Features

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

Complexity:

Explicit polynomials $\rightarrow n_{\text{samples}} * n_{\text{features}}^d$

Kernel trick $\rightarrow n_{\text{samples}} * n_{\text{samples}} * n_{\text{features}}$

For a single feature:

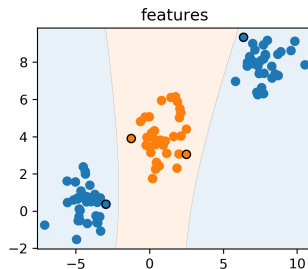
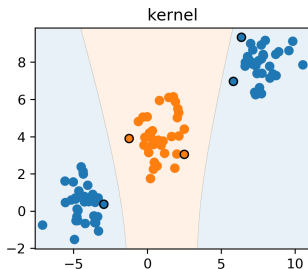
$$(x^2, \sqrt{2}x, 1)^T (x'^2, \sqrt{2}x', 1) = x^2 x'^2 + 2xx' + 1 = (xx' + 1)^2$$

Poly kernels vs explicit features

```

1 poly = PolynomialFeatures(include_bias=False)
2 X_poly = poly.fit_transform(X)
3 print(X.shape, X_poly.shape)
4 print(poly.get_feature_names())
5
6 ((100, 2), (100, 5))
7 ['x0', 'x1', 'x0^2', 'x0 x1', 'x1^2']

```



Understanding Dual Coefficients

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

```
1 linear_svm.coef_  
2  
3 array([[0.139, 0.06, -0.201, 0.048, 0.019]])
```

$$y = \text{sign}(-0.03\phi(\mathbf{x}_1)^T \phi(\mathbf{x}) - 0.003\phi(\mathbf{x}_{26})^T \phi(\mathbf{x}) + 0.003\phi(\mathbf{x}_{42})^T \phi(\mathbf{x}) + 0.03\phi(\mathbf{x}_{62})^T \phi(\mathbf{x}))$$

```
1 linear_svm.dual_coef_  
2  
3 array([[ -0.03, -0.003, 0.003, 0.03]])  
4  
5 linear_svm.support_  
6  
7 array([1,26,42,62], dtype=int32)
```

With Kernel

$$y = \text{sign} \left(\sum_i^n \alpha_i k(\mathbf{x}_i, \mathbf{x}) \right)$$

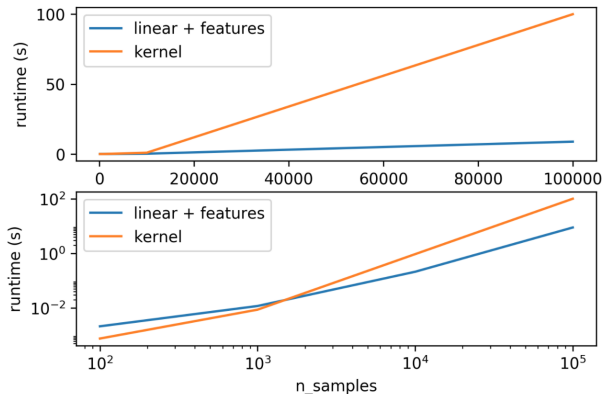
```

1 poly_svm.dual_coef_
2
3 array([[ -0.057,  -0.   ,  -0.012,  0.008,  0.062]])
4
5 poly_svm.support_
6
7 array([1,26,41,42,62], dtype=int32)

```

$$y = \text{sign}(-0.057(\mathbf{x}_1^T \mathbf{x} + 1)^2 - 0.012(\mathbf{x}_{41}^T \mathbf{x} + 1)^2 + 0.008(\mathbf{x}_{42}^T \mathbf{x} + 1)^2 + 0.062(\mathbf{x}_{62}^T \mathbf{x} + 1)^2)$$

Runtime Considerations



Kernels in Practice

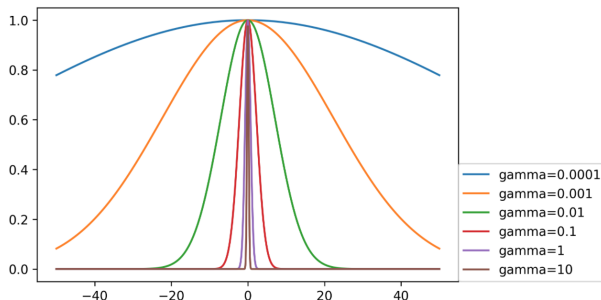
- Dual coefficients less interpretable
- Long runtime for “large” datasets (100k samples)
- As a rule of thumb, try linear kernel first (remember ‘LinearSVC’ is much faster than ‘SVC(kernel=linear)’)
- If the training data is not so large, try Gaussian RBF kernel

Preprocessing

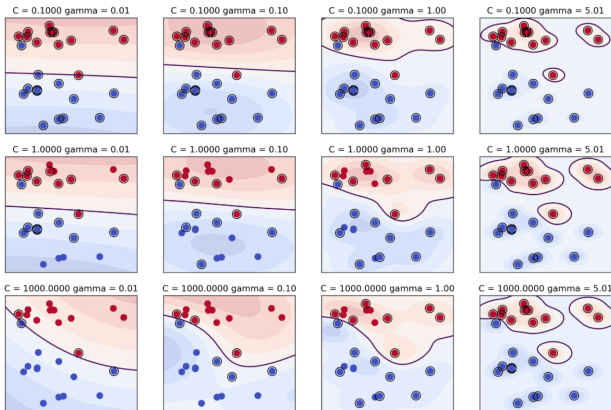
- Kernel use inner products or distances.
- StandardScaler or MinMaxScaler
- Gamma parameter in RBF directly relates to scaling of data and `n_features` – the default is $1/(X.\text{var}() * n_features)$

Parameters for RBF Kernels

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

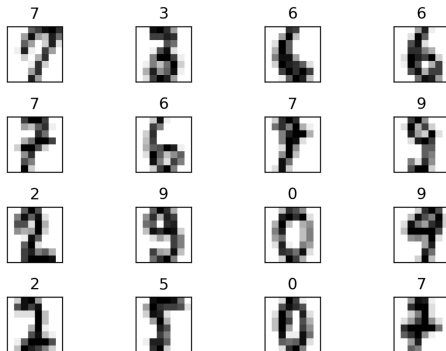


Parameters for RBF Kernels



MNIST Example

```
1 from sklearn.datasets import load_digits
2
3 digits = load_digits()
```



Scaling and Default Params

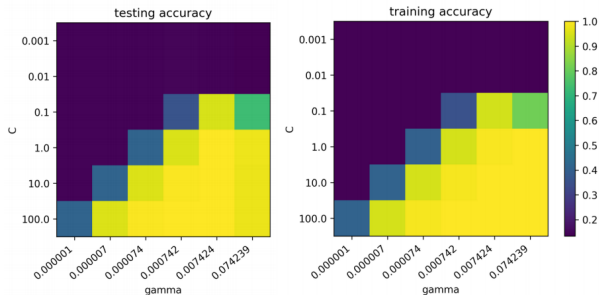
```
1 # gamma : {'scale', 'auto'} or float, optional (default='scale')
2 # Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.
3 # if gamma='scale' (default) is passed then it uses 1 / (n_features * X.var())
4 # as value of gamma
5 # if 'auto', uses 1 / n_features.
6
7 print('auto', np.mean(cross_val_score(SVC(gamma='auto'), X_train, y_train, cv=10)))
8 print('scale', np.mean(cross_val_score(SVC(gamma='scale'), X_train, y_train, cv=10)))
9 scaled_svc = make_pipeline(StandardScaler(), SVC())
10 print('pipe', np.mean(cross_val_score(scaled_svc, X_train, y_train, cv=10)))
11
12
13 auto 0.563
14 scale 0.987
15 pipe 0.977
16
17 gamma = (1. / (X_train.shape[1] * X_train.var()))
18 print(np.mean(cross_val_score(SVC(gamma=gamma), X_train, y_train, cv=10)))
19
20 0.987
```

Grid-Searching Parameters

```

1 param_grid = {'svc_C': np.logspace(-3, 2, 6),
2               'svc_gamma': np.logspace(-3, 2, 6) / X_train.shape[0]}
3 param_grid
4
5 {'svc_C': array([0.001, 0.01, 0.1, 1., 10., 100.]),
6  'svc_gamma': array([ 0.000001, 0.000007, 0.000074,
7                     0.000742, 0.007424, 0.074239])}
8
9 grid = GridSearchCV(scaled_svc, param_grid=param_grid, cv=10)
10 grid.fit(X_train, y_train)

```



Summary

- Logistic Regression and Linear SVM differ from each other by their loss functions