

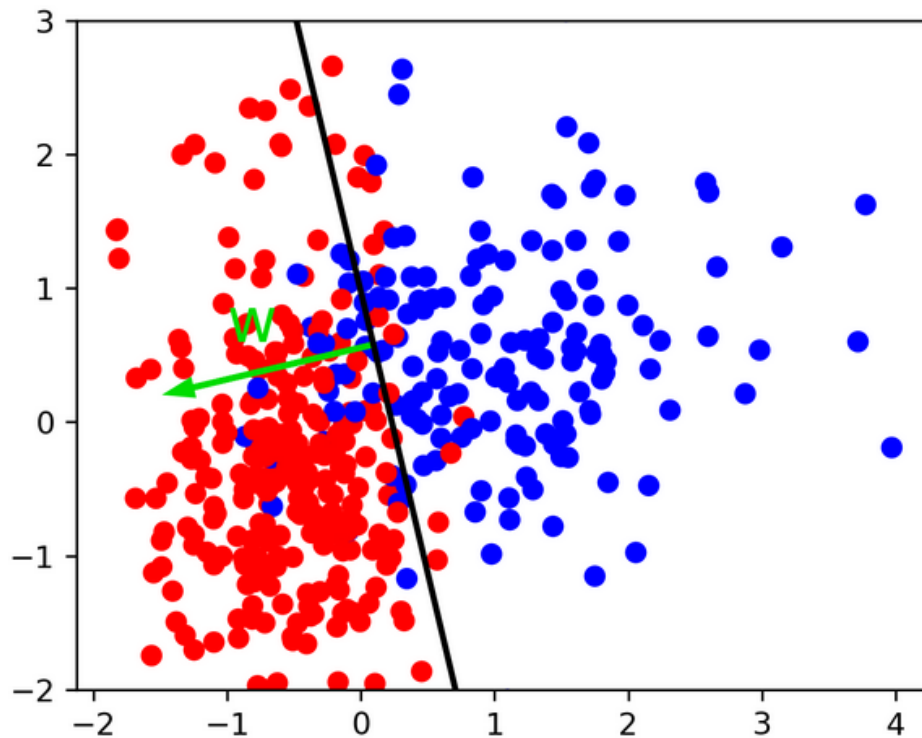
MACHINE LEARNING AVANZATO DA ZERO

ANTONIO DI CECCO - SCHOOL OF AI

Modelli Lineari per la classificazione e SVM

Modelli lineari di classificazione binaria

- Cane o gatto?
- Obiettivi discreti



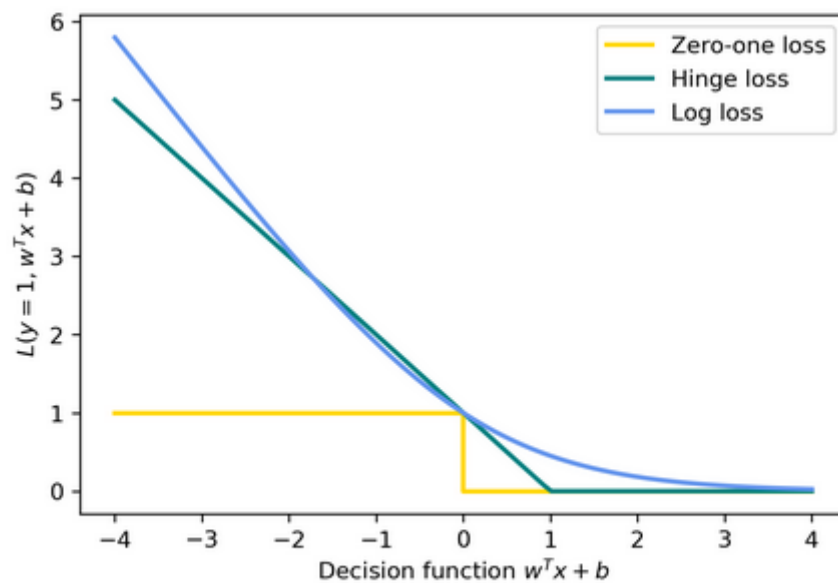
$$\hat{y} = \text{sign}(w^T \mathbf{x} + b) = \text{sign} \left(\sum_i w_i x_i + b \right)$$

- è il nostro modello lineare con decision boundary

Come scelgo la loss?

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

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



- La zero-one loss non è convessa
- il problema di ottimizzazione sarebbe NP- complete
- Log loss e Hinge loss approssimano la zero-one loss dall'alto
- la Log loss nasce da considerazioni statistiche
- la Hinge loss definisce un margine: l'errore cresce linearmente tanto più mi allontano dal bordo del margine

Logistic Regression

Logistic Regression

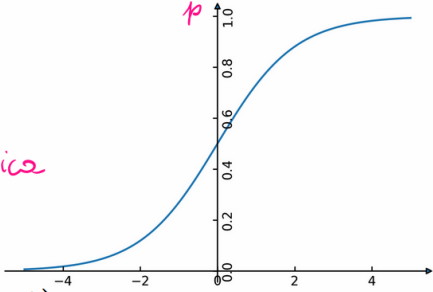
1:10 odds

$$\log\left(\frac{p(y=1|x)}{p(y=-1|x)}\right) = w^T \mathbf{x} + b$$

decision

$$p(y=1|\mathbf{x}) = \frac{1}{1 + e^{-w^T \mathbf{x} - b}}$$

Logistica



$$\min_{w \in \mathbb{R}^p, b \in \mathbb{R}} \sum_{i=1}^n \log(\exp(-y_i(w^T \mathbf{x}_i + b)) + 1)$$

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

$\log \prod_i p_i = \sum_i \log p_i = \text{MAXIMA}$
 $\sum_i \log \frac{1}{p_i} = \text{MINIMA}$
 Versinghiana

- si chiama Regression ma è una Classification
- quello che "regredisco" è la probabilità attesa della classe o più precisamente le **odds**
- dalla probabilità attesa ottengo la classe target con una soglia (e.g. il 50%)
- per la logistica la soglia è proprio con l'argomento a 0

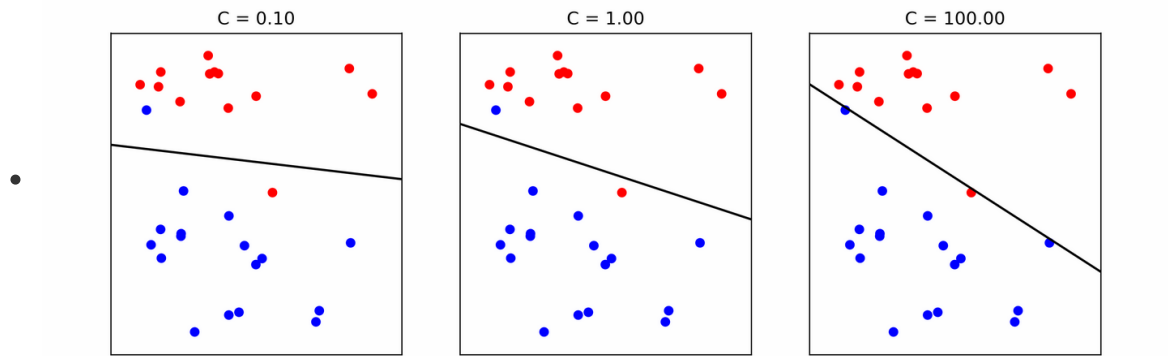
Regularizzo la Logistic Regression

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

$$\min_{w \in \mathbb{R}^p, b \in \mathbb{R}} C \sum_{i=1}^n \log(\exp(-y_i(w^T \mathbf{x}_i + b)) + 1) + ||w||_1$$

- C è l'inverso di alpha (o di alpha / n_samples)

Effect of regularization

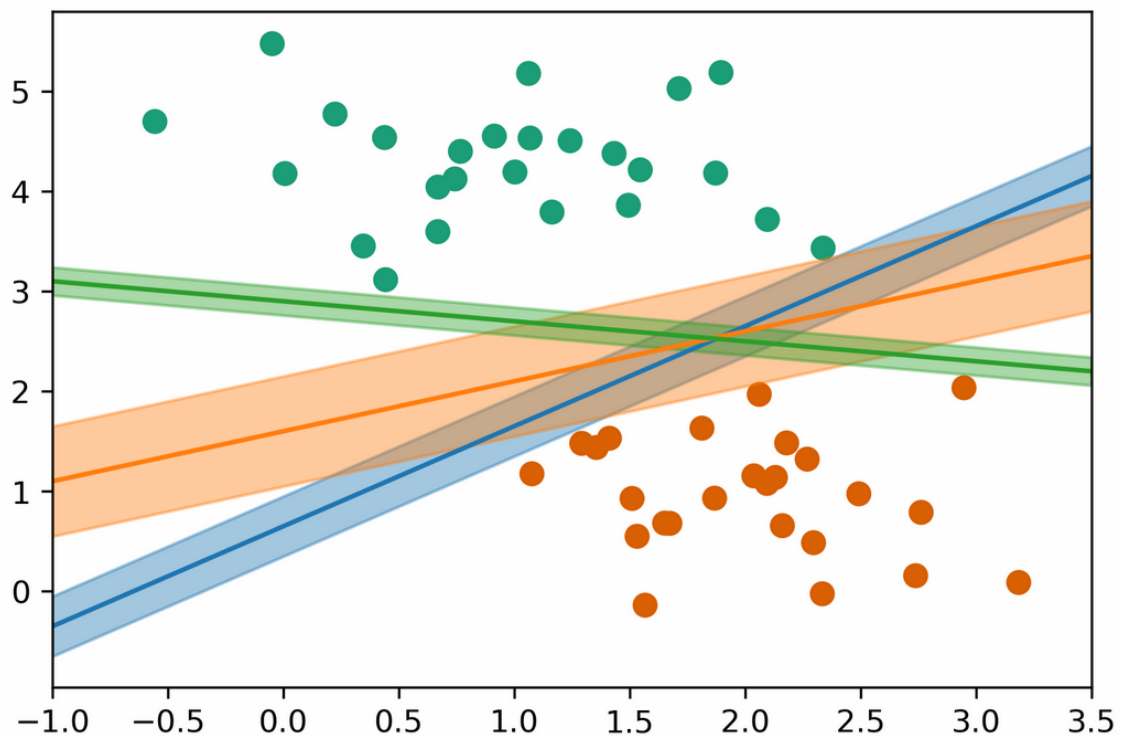


- Small C (a lot of regularization) limits the influence of individual points!

La regressione logistica è straordinariamente buona

- molto usata in industri per la sua semplicità
 - **interpretabilità**
 - insospettabilmente accurata perché non presuppone che i residui siano gaussiani a differenza della regressione lineare
-

Vettori di supporto e massimo margine (SVM)

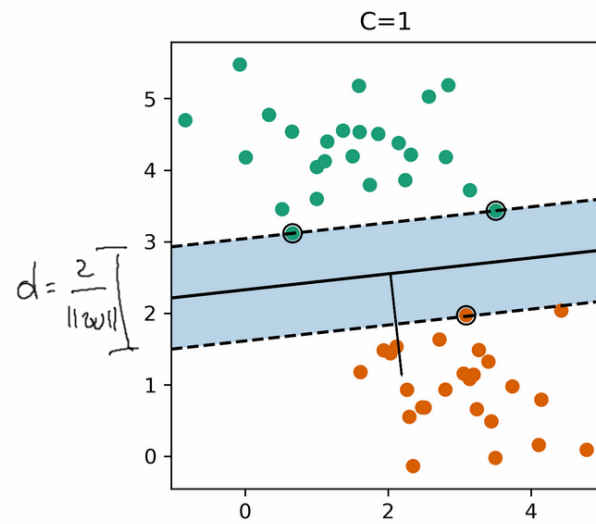
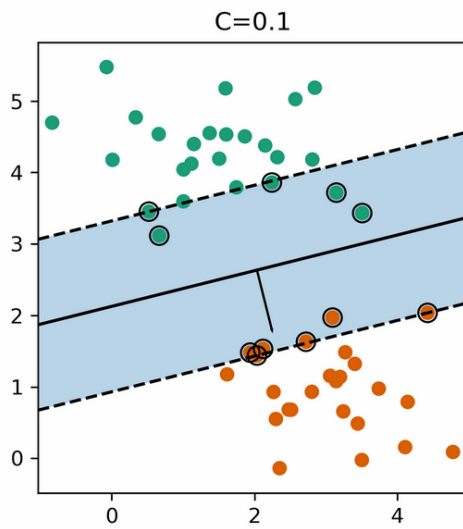


$$\min_{w \in \mathbb{R}^p, b \in \mathbb{R}} C \sum_{i=1}^n \max(0, 1 - y_i(w^T \mathbf{x} + b)) + \|w\|_2^2$$

Within margin $\Leftrightarrow y_i(w^T x + b) < 1$

Smaller $w \Rightarrow$ larger margin $\quad \|\mathcal{W}\| = \frac{2}{\alpha}$

(soft margin) linear SVM



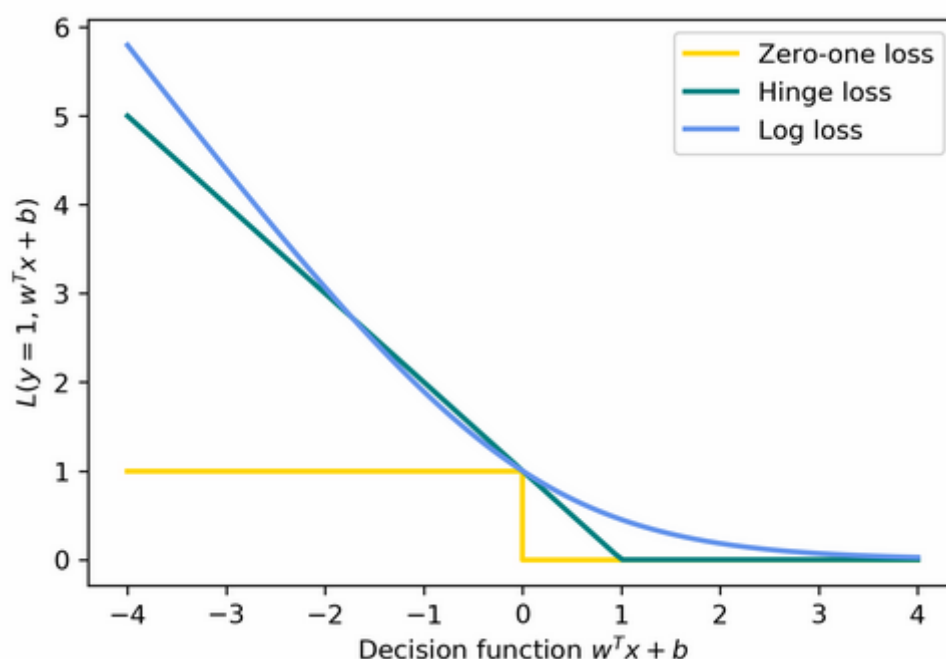
- Tutti i vettori nella banda sono vettori di supporto
- più dati utilizzati
- margine più ampio

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

$$\min_{w \in \mathbb{R}^p, b \in \mathbb{R}} C \sum_{i=1}^n \max(0, 1 - y_i(w^T \mathbf{x}_i + b)) + \|w\|_1$$

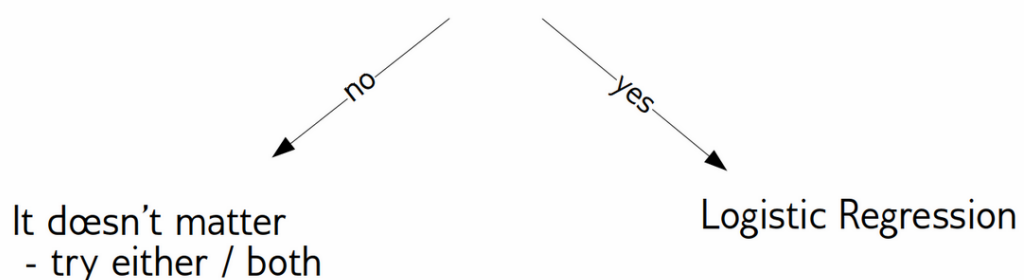
Logistic Regression vs SVM

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



SVM or Logistic Regression?

Do you need probability estimates?



- Need compact model or believe solution is sparse? Use L1

- la Logistic Regression nasce con la probabilità in mente
- la SVM deve essere calibrata ma come vedremo è universale

Multiclass classification

Reduction to Binary Classification

One vs Rest

One vs One

One vs Rest

One Vs Rest

For 4 classes:

1v{2,3,4}, 2v{1,3,4}, 3v{1,2,4}, 4v{1,2,3}

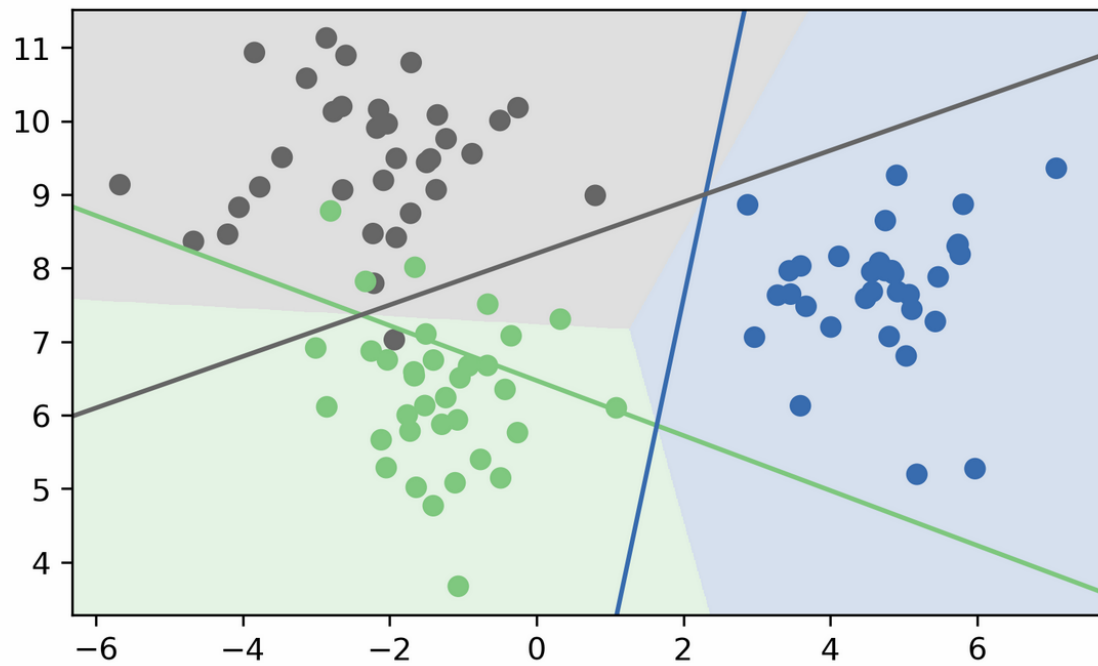
In general:

n binary classifiers - each on all data

Prediction with One Vs Rest

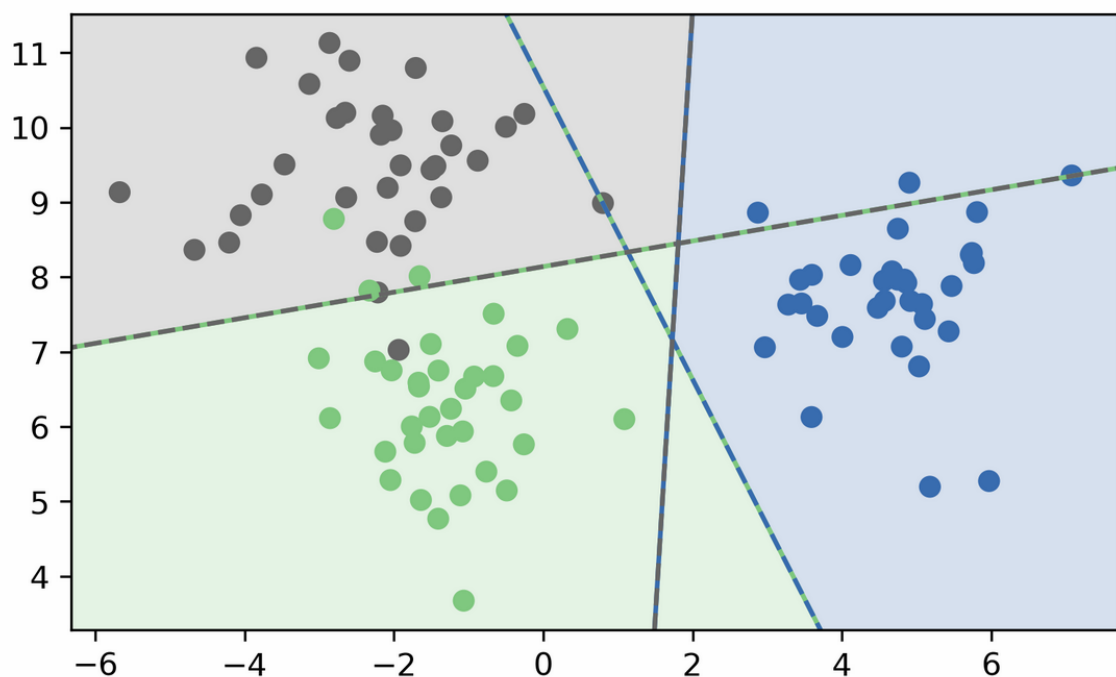
"Class with highest score"

$$\hat{y} = \arg \max_{i \in Y} \mathbf{w}_i^T \mathbf{x} + b_i$$



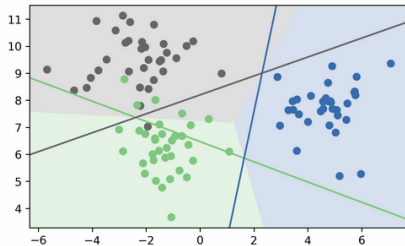
One vs One

- 1v2, 1v3, 1v4, 2v3, 2v4, 3v4
- $n * (n-1) / 2$ classificatori binari - ognuno sulla propria frazione dei dati
- si "Vota per il numero più alto di positivi"
- Classifica usando tutti i classificatori.
- Si conta quanto spesso ogni classe viene predetta.
- L'output è la classe più frequentemente predetta.
- anche questa è solo un'euristica



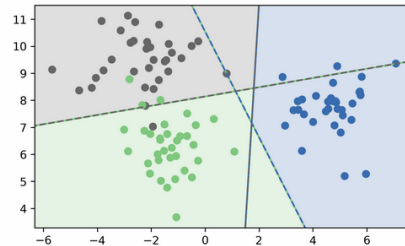
One vs Rest

- n_classes classifiers
- trained on imbalanced datasets of original size
- Retains some uncertainty?



One vs One

- n_classes * (n_classes - 1)/2 classifiers
- trained on balanced subsets
- No uncertainty propagated



Multinomial Logistic Regression

Probabilistic multi-class model:

$$p(y = i|x) = \frac{e^{\mathbf{w}_i^T \mathbf{x} + b_i}}{\sum_{j=1}^k e^{\mathbf{w}_j^T \mathbf{x} + b_j}}$$

$$\min_{w \in \mathbb{R}^{pk}, b \in \mathbb{R}^k} - \sum_{i=1}^n \log(p(y = y_i | x_i, w, b))$$

$$\hat{y} = \arg \max_{i=1, \dots, k} \mathbf{w}_i^T \mathbf{x} + b_i$$

- Same prediction rule as OvR !

In scikit-learn

- OvO: only SVC
- OvR: default for all linear models except for logistic regression
- `LogisticRegression(multi_class='auto')`
- `clf.decision_function = $w^T x + b$`
- `logreg.predict_proba`
- `SVC(probability=True)` not great

↑ USA LA CALIBRAZIONE

Multi-Class in Practice

OvR and multinomial LogReg produce one coef per class:

```
from sklearn.datasets import load_iris
iris = load_iris()
X, y = iris.data, iris.target
print(X.shape)
print(np.bincount(y))
```

```
(150, 4)
[50 50 50]
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
```

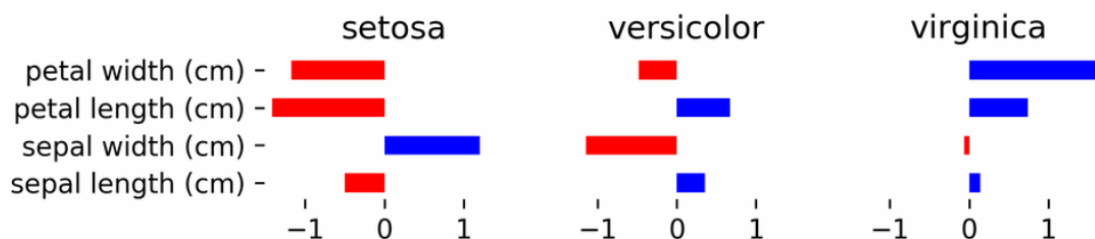
```
logreg = LogisticRegression(multi_class="multinomial", solver="lbfgs").fit(X, y)
linearsvm = LinearSVC().fit(X, y)
print(logreg.coef_.shape)
print(linearsvm.coef_.shape)
```

```
(3, 4)
(3, 4)
```

SVC would produce the same shape, but with different semantics!

logreg.coef_

```
array([[ -0.42339232,  0.96169329, -2.51946669, -1.0860205 ],
       [ 0.53411332, -0.31794321, -0.20537377, -0.93961515],
       [-0.11072101, -0.64375008,  2.72484045,  2.02563566]])
```



(after centering data, without intercept)

Computational Considerations

(for all linear models)

- Don't use SVC(kernel='linear'), use LinearSVC OvO
- For $n_{\text{features}} \gg n_{\text{samples}}$: Lars (or LassoLars) instead of Lasso. OvR
- For small n_{samples} (<10.000?), don't worry.
- LinearSVC, LogisticRegression: dual=False if $n_{\text{samples}} \gg n_{\text{features}}$
- LogisticRegression(solver="sag") for n_{samples} large.
- Stochastic Gradient Descent for n_{samples} really large

Kernel SVMs

Motivazioni

- Transizione dai modelli lineari ai più potenti modelli non lineari.
 - I modelli lineari non sono abbastanza complessi
 - Sono sensibili agli outlier
 - Mantiene la convessità (facile da ottimizzare).
 - Generalizza il concetto di feature engineering.
-

Richiamo di Linear SVM

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

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

Ottimizzazione duale dei modelli lineari

- Optimization Theory

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

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

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

$$\alpha_i \leq C$$

- I pesi sono funzione lineare dei nostri dati (dei vettori di supporto nelle SVM)
- il modello diventa combinazione lineare dei prodotti scalari

Introduciamo i Kernel e il Kernel Trick

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

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

Kernel Trick

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

- ϕ è la nostra *trasformazione* ad esempio il logaritmo applicato alle feature di una serie storica.
- ϕ può essere vista come un estrattore di feature
- ϕ si chiama "*rappresentazione*"

Esempi di kernel

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

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

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

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

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

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

Il kernel trick con feature polinomiali

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

Primal vs Dual Optimization

Explicit polynomials → compute on $n_{\text{samples}} * n_{\text{features}} ** d$

Kernel trick → compute on kernel matrix of shape $n_{\text{samples}} * n_{\text{samples}}$

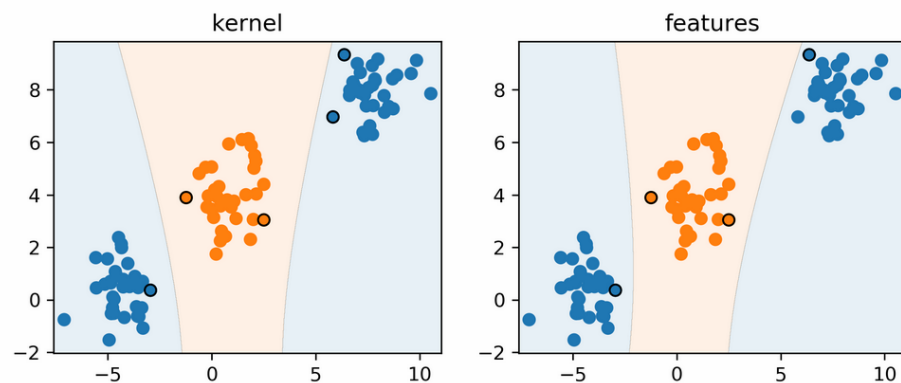
For a single feature:

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

- non è la stessa cosa ma ci si avvicina

```
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']
```



Capire i coefficienti duali

```
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)
```

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

con il Kernel

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

```
poly_svm.dual_coef_
```

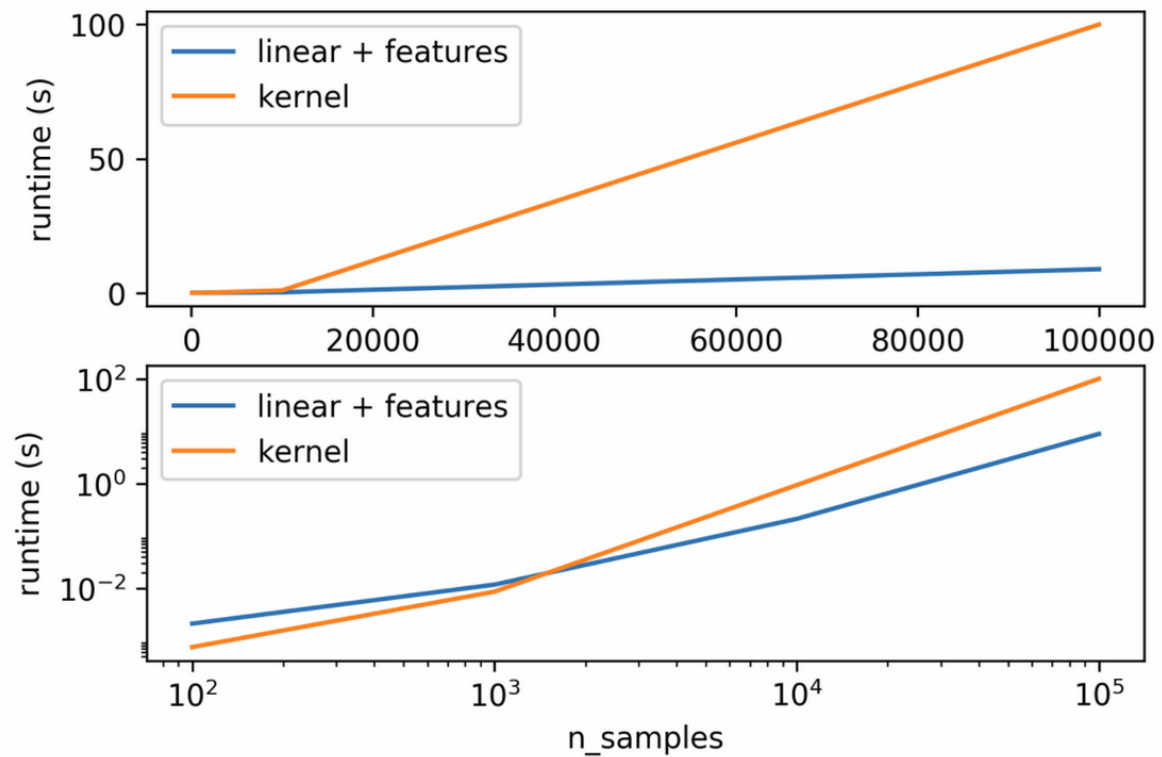
```
# array([[ -0.057,  -0. ,  -0.012,  0.008,  0.062]])
```

```
poly_svm.support_
```

```
# array([1,26,41,42,62], dtype=int32)
```

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

Runtime Considerations



Kernels in Pratica

- i Dual coefficient sono meno interpretabili
- Lente
- La vera forza è negli spazi infinito dimensionali: rbf!
- Rbf è “un kernel universale” - può apprendere tutto (e overfittare tutto)

Vecchia diatriba: SVM o NN ?

- vedremo che le NN sono più espressive permettendo un apprendimento gerarchico
- recentemente è stato dimostrato che tutti i modelli addestrati con il gradient descent sono surrogabili con una SVM

Every Model Learned by Gradient Descent Is Approximately a Kernel Machine

Pedro Domingos

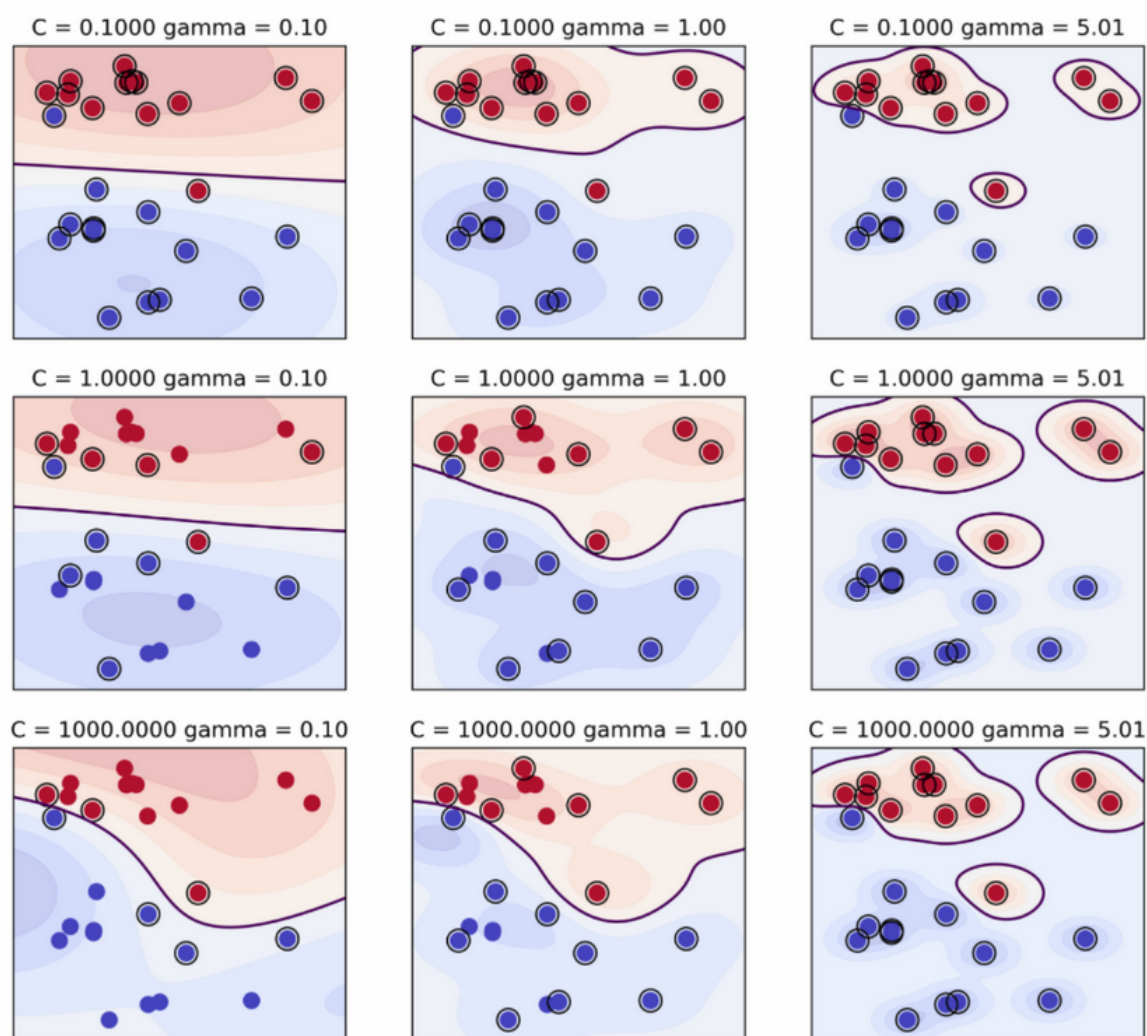
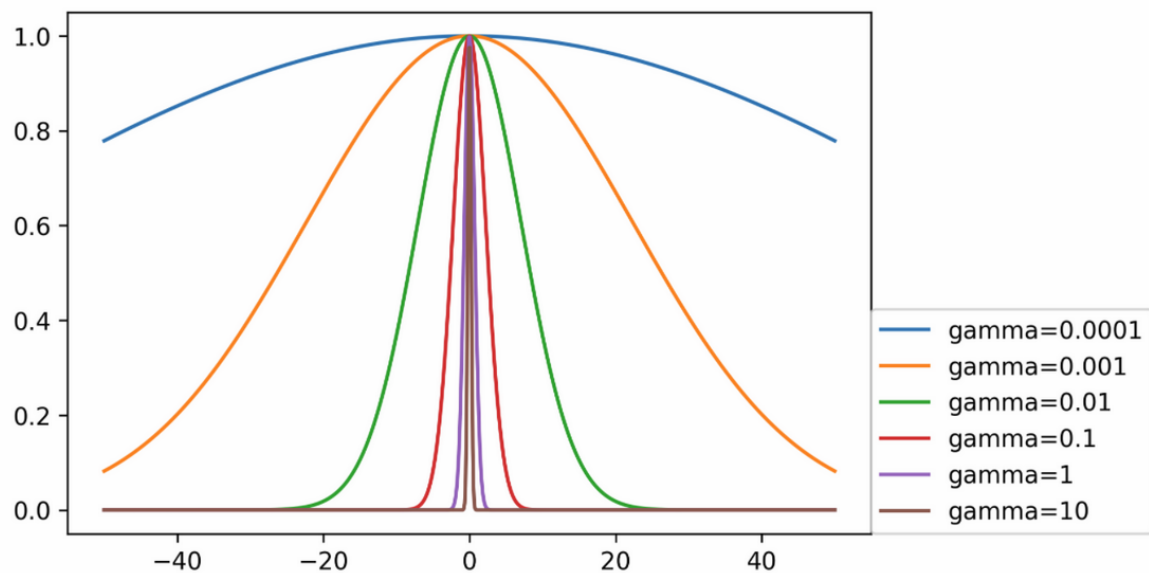
Deep learning's successes are often attributed to its ability to automatically discover new representations of the data, rather than relying on handcrafted features like other learning methods. We show, however, that deep networks learned by the standard gradient descent algorithm are in fact mathematically approximately equivalent to kernel machines, a learning method that simply memorizes the data and uses it directly for prediction via a similarity function (the kernel). This greatly enhances the interpretability of deep network weights, by elucidating that they are effectively a superposition of the training examples. The network architecture incorporates knowledge of the target function into the kernel. This improved understanding should lead to better learning algorithms.

Preprocessing

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

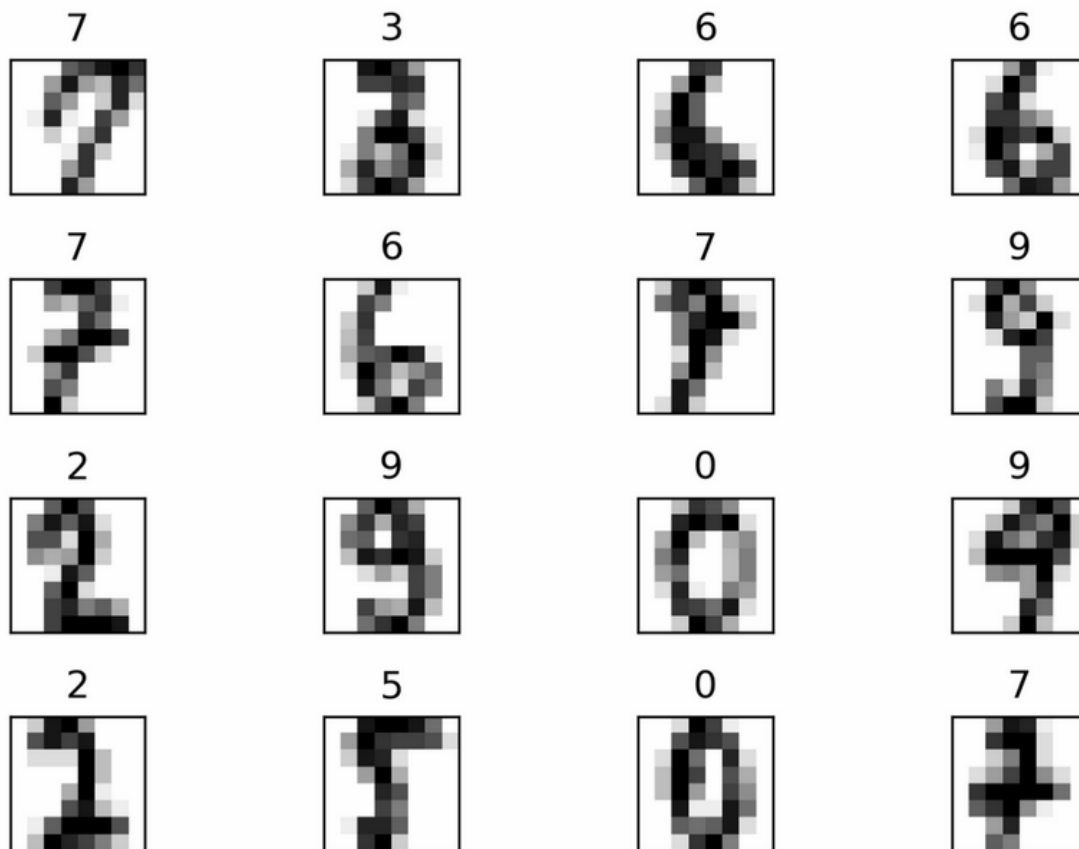
Parameters for RBF Kernels

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



[Homework]: classificazione del digit dataset

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



Scaling and Parametri di default

gamma : {'scale', 'auto'} or float, optional (default='scale')
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.
if gamma='scale' (default) is passed then it uses $1 / (n_features * X.var())$
as value of gamma
if 'auto', uses $1 / n_features$.

```
rint('auto', np.mean(cross_val_score(SVC(gamma='auto'), X_train, y_train, cv=10)))
rint('scale', np.mean(cross_val_score(SVC(gamma='scale'), X_train, y_train, cv=10)))
scaled_svc = make_pipeline(StandardScaler(), SVC())
rint('pipe', np.mean(cross_val_score(scaled_svc, X_train, y_train, cv=10)))
```

```
auto 0.563
scale 0.987
pipe 0.977
```

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

- gamma scala con il numero di feature
- gamma scala con la varianza

Grid-Searching dei parametri

```
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([ 0.0000001, 0.0000007, 0.0000074,
                    0.000742, 0.007424, 0.074239])}
```

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

