Machine Learning

Università Roma Tre
Dipartimento di Ingegneria

Anno Accademico 2021 - 2022

Esercitazione: Regressione e Classificazione (Ex04)

Sommario

- Esercizi su Linear models per la classificazione

Richiami: Linear models per la classificazione

- I modelli lineari possono essere impiegati per la classificazione con *decision boundary* che rappresento linee, piani o iperpiani.
- Nella logisitic regression si impiega un modello lineare tradizionale il cui output è valutato da una funzione *logistic* (*sigmoid function*) che restituisce un valore in [0,1] ed indica la probabilità di appartenenza ad una certa classe (> 0.5) o meno (< 0.5).
- Il *gradient descent* è impiegato per minimizzare la funzione di costo.
- I due algoritmi di classificazione lineari più famosi sono la *logistic regression,* e i *linear support vector machine* (o Linear SVM) che vedremo in seguito.
- Scikit-learn fornisce la classe *linear_model.LogisticRegression* e *svm.LinearSVC* che implementa i due algoritmi.

Linear models per la classificazione: forge dataset

- Esercizio: impiega il forge dataset e la LogisticRegression per la classificazione. Valuta le performance.

```
from sklearn.linear_model import LogisticRegression

X, y = mglearn.datasets.make_forge()


...
```

Linear models per la classificazione: forge dataset

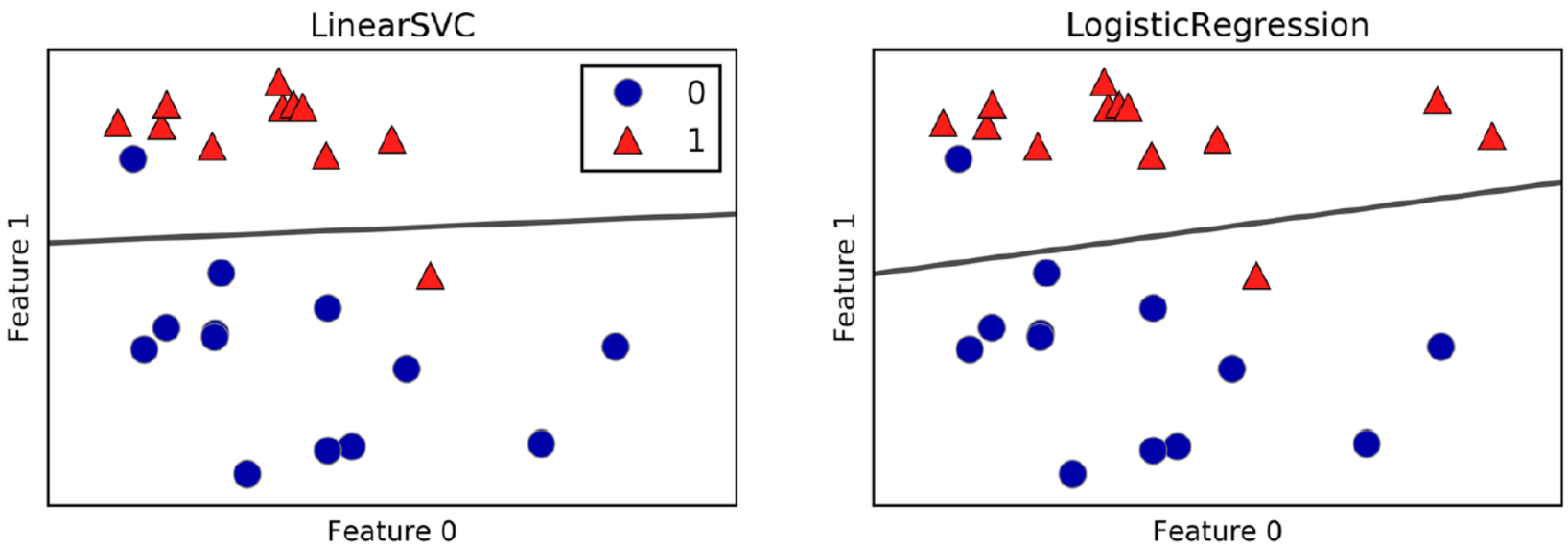
- Esercizio: impiega il forge dataset e la LogisticRegression per la classificazione. Valuta le performance.

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
X, y = mglearn.datasets.make_forge()

fig, axes = plt.subplots(1, 2, figsize=(10, 3))

for model, ax in zip([LinearSVC(), LogisticRegression()], axes):
    clf = model.fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=False, eps=0.5,
        ax=ax, alpha=.7)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{}".format(clf.__class__.__name__))
    ax.set_xlabel("Feature 0")
    ax.set_ylabel("Feature 1")
axes[0].legend()
```

- Nota: Impieghiamo entrambe le implementazioni, anche se l'algoritmo SVM lo vedremo in dettaglio in seguito.

Linear models per la classificazione: forge dataset

- In questo dataset la decision boundary è rappresentata da una retta.

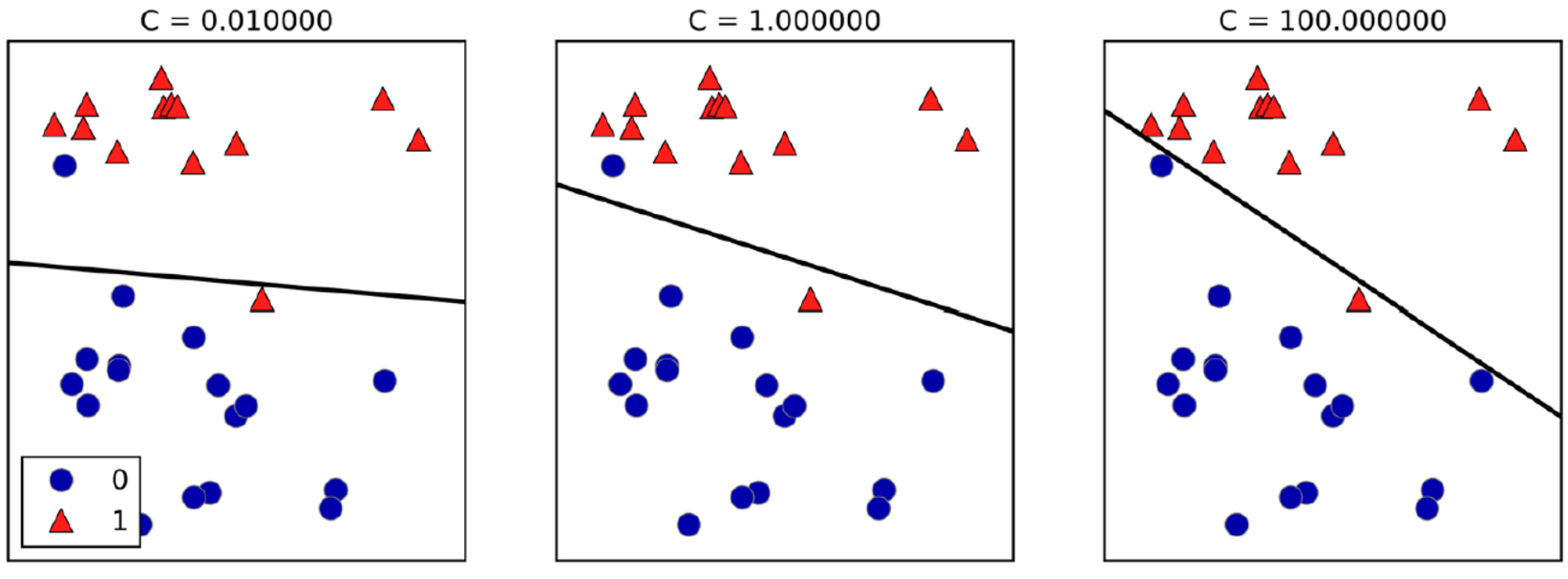


Scikit-learn: Logistic regression

- In Scikit-learn, la Logistic regression impiega la *L2* di default.
- Il parametro *penalty* specifica quale regolarizzazione impiegare {'l1', 'l2', 'elasticnet', 'none'}.
- Il parametro *C* indica il peso della regolarizzazione (valori bassi rappresentano una regolarizzazione maggiore), il default è C=1.0

Richiami: Linear models per la classificazione

- Al variare di *C* si può notare l'effetto sul dataset considerato.
- Con alta regolarizzazione (C basso) il modello sbaglia a classificare 2 istanze cercando di considerare la "maggioranza" durante la scelta della decision boundary.
- Per valori più alti di C la retta si inclina dando più importanza ai 2 punti. Un punto rimane comunque non classificato, ed è impossibile considerarlo con una semplice linea retta.



LinearLogistic: Breast cancer dataset

- Esercizio: valuta la *linera logistic* nel caso del Breast cancer dataset, considerando valori di C pari a 0.01, 1 e 100. Commenta i risultati ottenuti in termini di accuracy e potenziali fenomeni di over o underfitting.

LinearLogistic: Breast cancer dataset

- Esercizio: valuta la linera logistic nel caso del Breast cancer dataset, considerando valori di C pari a 1, 100 e 0.01.

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, stratify=cancer.target, random_state=42)
logreg = LogisticRegression(C=1).fit(X_train, y_train)
print("Training set score: {:.3f}".format(logreg.score(X_train, y_train)))
print("Test set score: {:.3f}".format(logreg.score(X_test, y_test)))
```

```
Training set score: 0.953
Test set score: 0.958

Training set score: 0.972
Test set score: 0.965

Training set score: 0.934
Test set score: 0.930
```

- Per C=1 c'è un probabile underfitting. Per C=100 (modello più complesso) migliorano le performance. Per C=0.01 si incrementa l'underfitting iniziale.

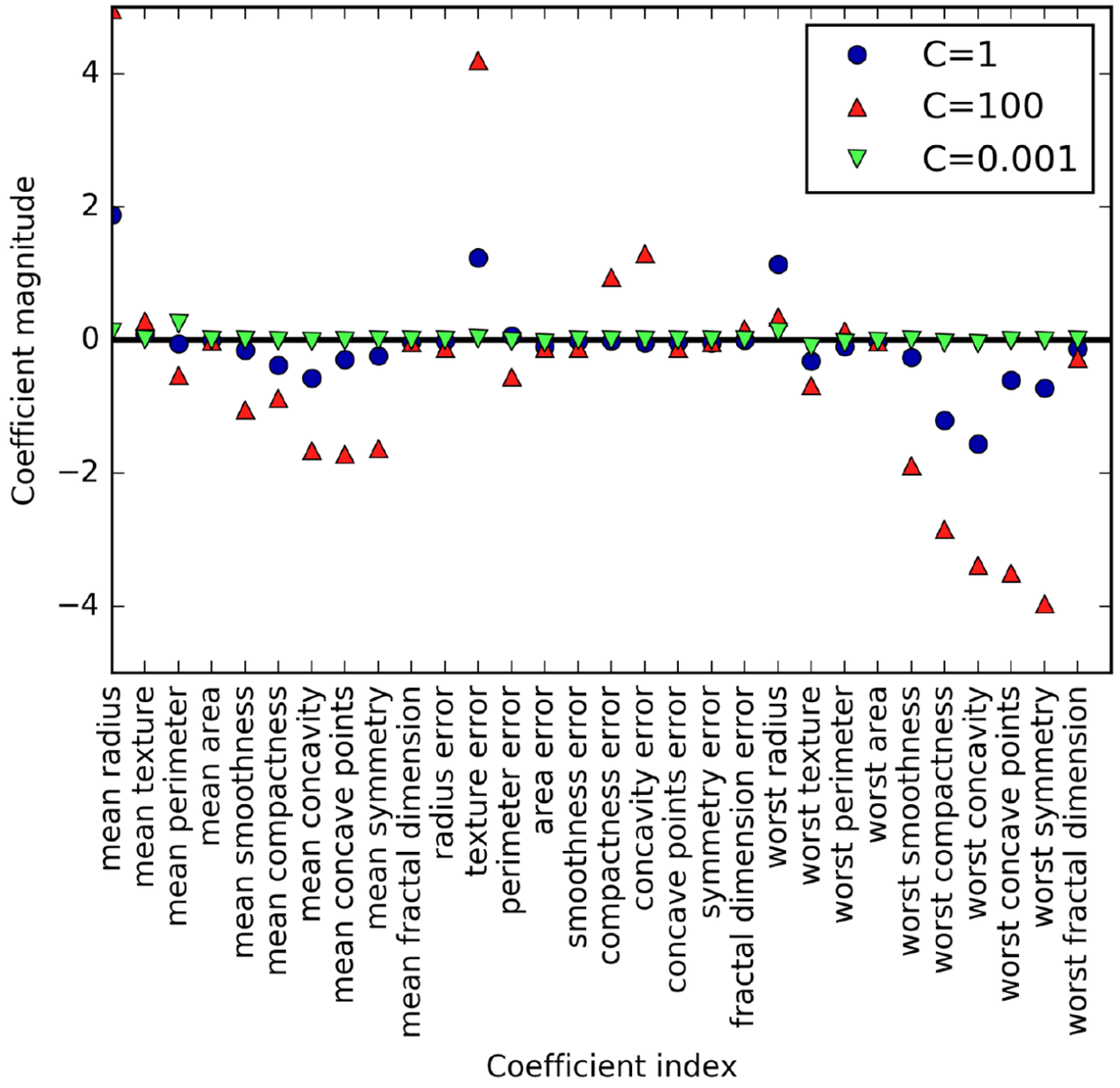
LinearLogistic: Breast cancer dataset

- Esercizio: visualizza i parametri del modello. Cosa ti aspetti?

```
plt.plot(logreg.coef_.T, 'o', label="C=1")
plt.plot(logreg100.coef_.T, '^', label="C=100")
plt.plot(logreg001.coef_.T, 'v', label="C=0.001")
plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
plt.hlines(0, 0, cancer.data.shape[1])
plt.ylim(-5, 5)
plt.xlabel("Coefficient index")
plt.ylabel("Coefficient magnitude")
plt.legend()
```

LinearLogistic: Breast cancer dataset

- Regolarizzazioni alte spingono i parametri a 0.
- In alcuni casi (*mean perimeter*), per C=100 e C=1 il coefficiente è negativo, positivo per C=0.001.
- **Attenzione:** È sbagliato pensare che i parametri suggeriscano quali features determinino direttamente la classe (tumore maligno o meno). L'importanza della feature dipende strettamente dal modello preso in considerazione.



LinearLogistic: Breast cancer dataset

- Esercizio: cerca di interpretare meglio l'importanza delle feature impiegando la L1 regularization.

LinearLogistic: Breast cancer dataset

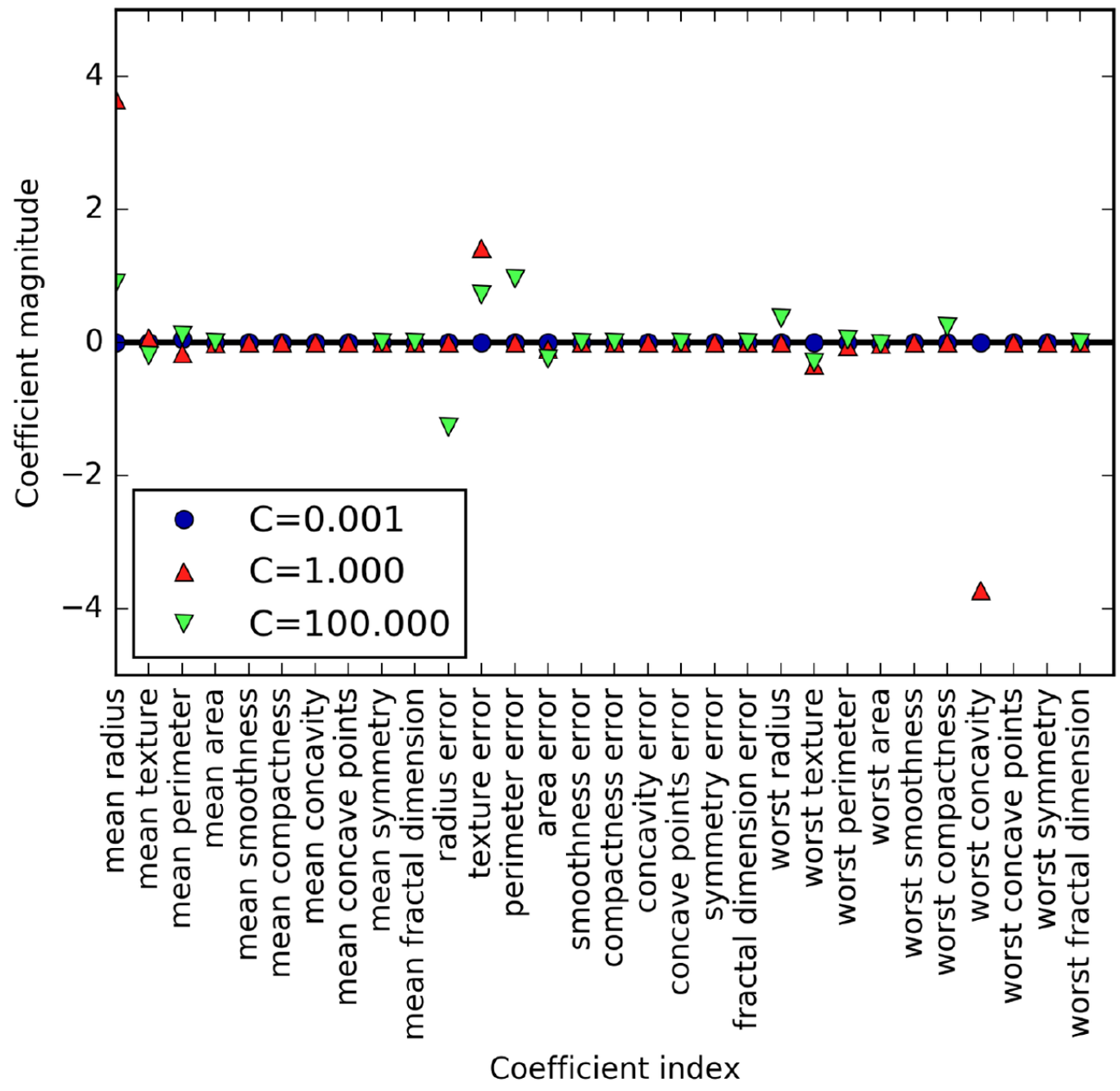
- Esercizio: cerca di interpretare meglio l'importanza delle feature impiegando la L1 regularization.

```
for C, marker in zip([0.001, 1, 100], ['o', '^', 'v']):
    lr_l1 = LogisticRegression(C=C, penalty="l1").fit(X_train, y_train)
    print("Training accuracy of l1 logreg with C={:.3f}: {:.2f}".format(
        C, lr_l1.score(X_train, y_train)))
    print("Test accuracy of l1 logreg with C={:.3f}: {:.2f}".format(
        C, lr_l1.score(X_test, y_test)))
    plt.plot(lr_l1.coef_.T, marker, label="C={:.3f}".format(C))

plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
plt.hlines(0, 0, cancer.data.shape[1])
plt.xlabel("Coefficient index")
plt.ylabel("Coefficient magnitude")

plt.ylim(-5, 5)
plt.legend(loc=3)
```

```
> Training accuracy of l1 logreg with C=0.001: 0.91
> Test accuracy of l1 logreg with C=0.001: 0.92
> Training accuracy of l1 logreg with C=1.000: 0.96
> Test accuracy of l1 logreg with C=1.000: 0.96
> Training accuracy of l1 logreg with C=100.000: 0.99
> Test accuracy of l1 logreg with C=100.000: 0.98
```

LinearLogistic: Breast cancer dataset

- L'effetto della L1 regularization dipende dal valore del parametro, in modo simile al caso della regressione.



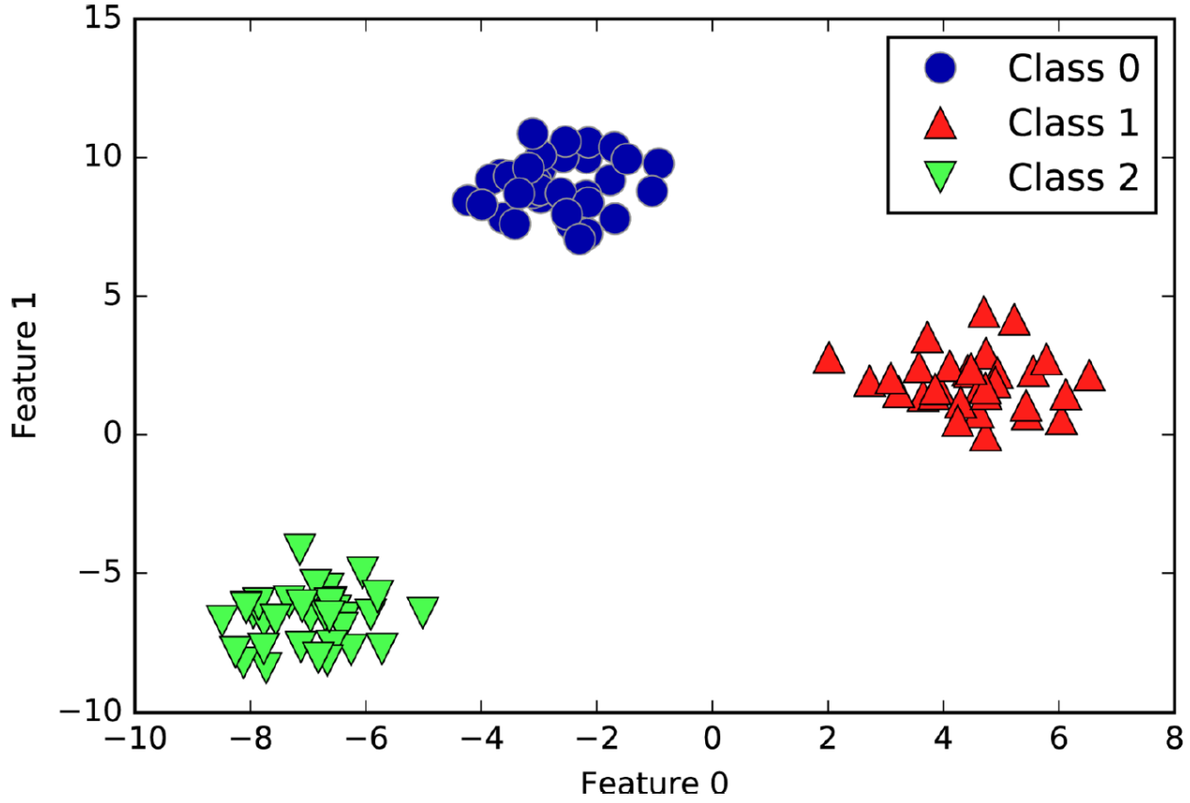
Linear models per la classificazione multiclass

- Alcuni modelli non si adattano facilmente al caso multiclass.
- Un approccio piuttosto semplice è il *one-vs-rest:* si creano vari modelli, dove ogni modello si addestra a riconoscere una specifica classe. Durante la predizione vengono valutati tutti i modelli e quello con score più alto determina la classe in output.
- Chiaramente si avranno un insieme di parametri da addestrare per ogni classe.

Linear models per la classificazione multiclass

- Esempio: creiamo un dataset con 2 features e 3 classi e impieghiamo il Linear SVM per la classificazione. Il dataset è creato seguendo una distribuzione gaussiana.

```
from sklearn.datasets import make_blobs
X, y = make_blobs(random_state=42)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
plt.legend(["Class 0", "Class 1", "Class 2"])
```



Linear models per la classificazione multiclass

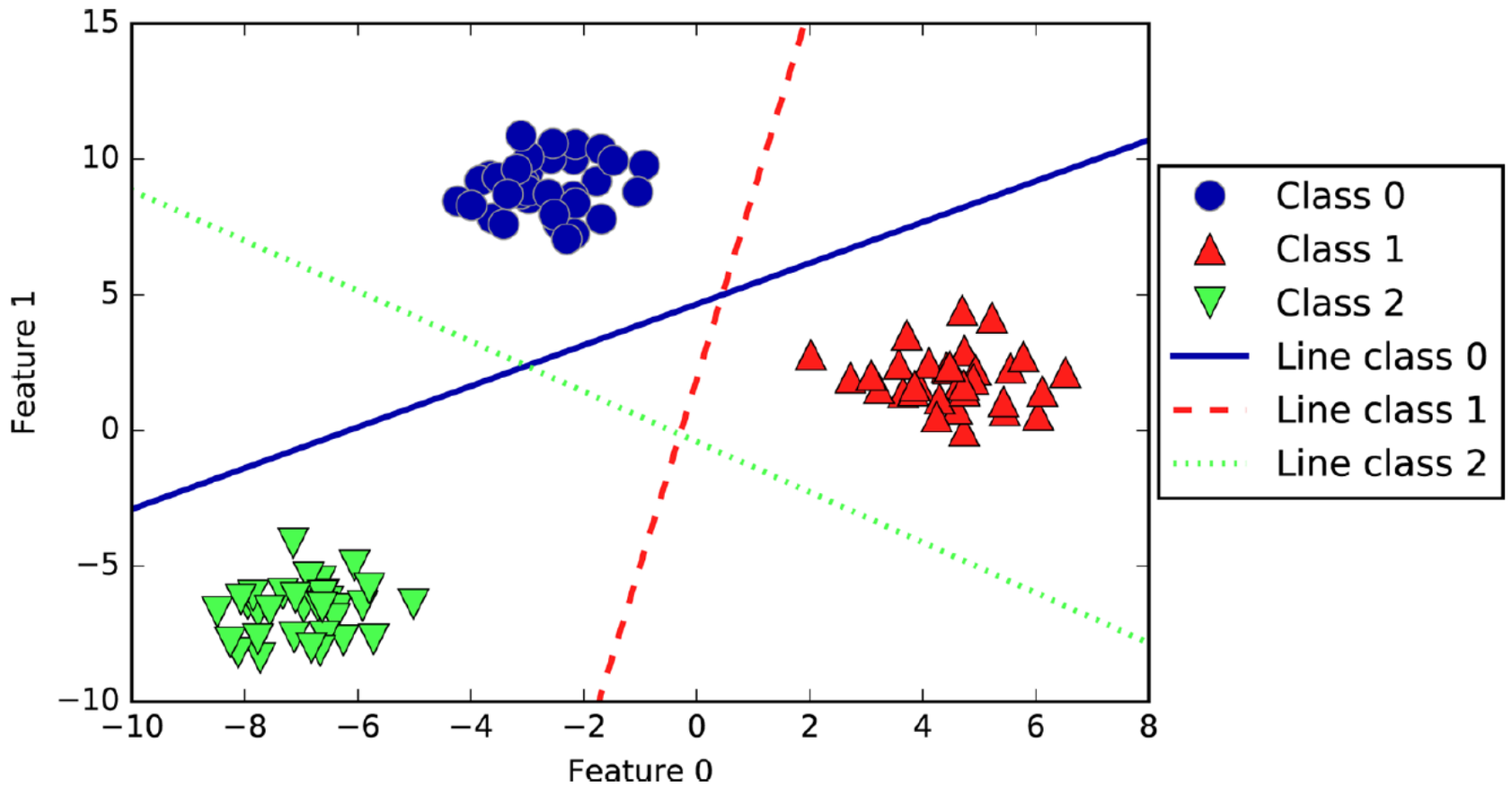
```
linear_svm = LinearSVC().fit(X, y)
print("Coefficient shape: ", linear_svm.coef_.shape)
print("Intercept shape: ", linear_svm.intercept_.shape)
> Coefficient shape: (3, 2)
>Intercept shape: (3,)
```

- Ogni riga di *_coef* rappresenta il vettore dei parametri per una delle 3 classi, e le colonne sono le 2 features. *intercept_* è un array 1d che memorizza l'intercetta per ogni classe.

```
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)
for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_,
['b', 'r', 'g']):
plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
plt.ylim(-10, 15)
plt.xlim(-10, 8)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
plt.legend(['Class 0', 'Class 1', 'Class 2', 'Line class 0', 'Line class 1',
'Line class 2'], loc=(1.01, 0.3))
```

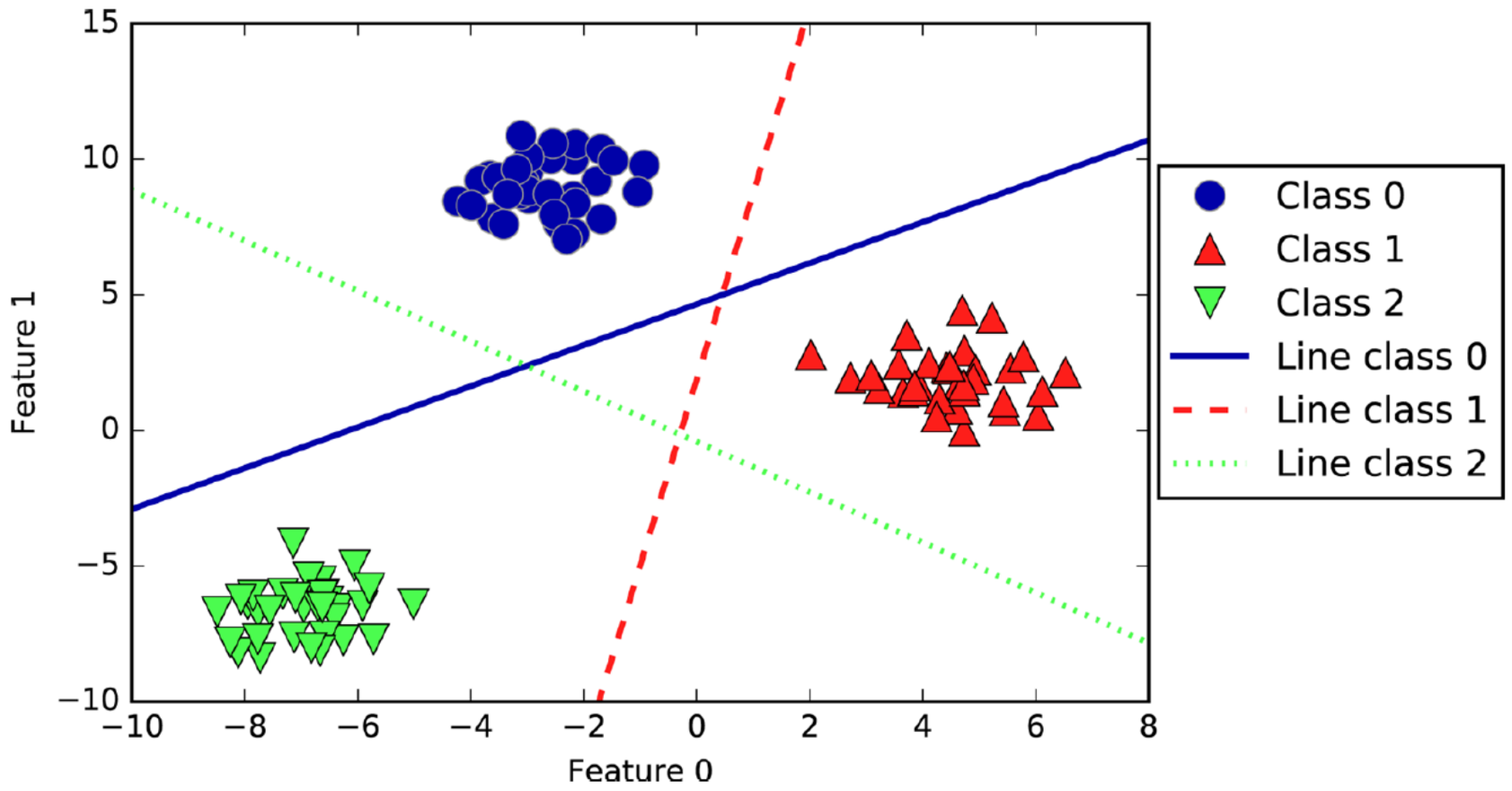
Linear models per la classificazione multiclass

- Ogni istanza etichettata con la classe 0 è al di sopra della *boundary* definita dal classificatore per la class 0. Le istanze delle altre classi al di sotto. Stessa cosa per la classe 1 e 2, e i relativi classificatori.
- Cosa succede se una istanza si trova nel triangolo centrale?



Linear models per la classificazione multiclass

- Cosa succede se una istanza si trova nel triangolo centrale?
  - La classe associata corrisponde alla linea più vicina.

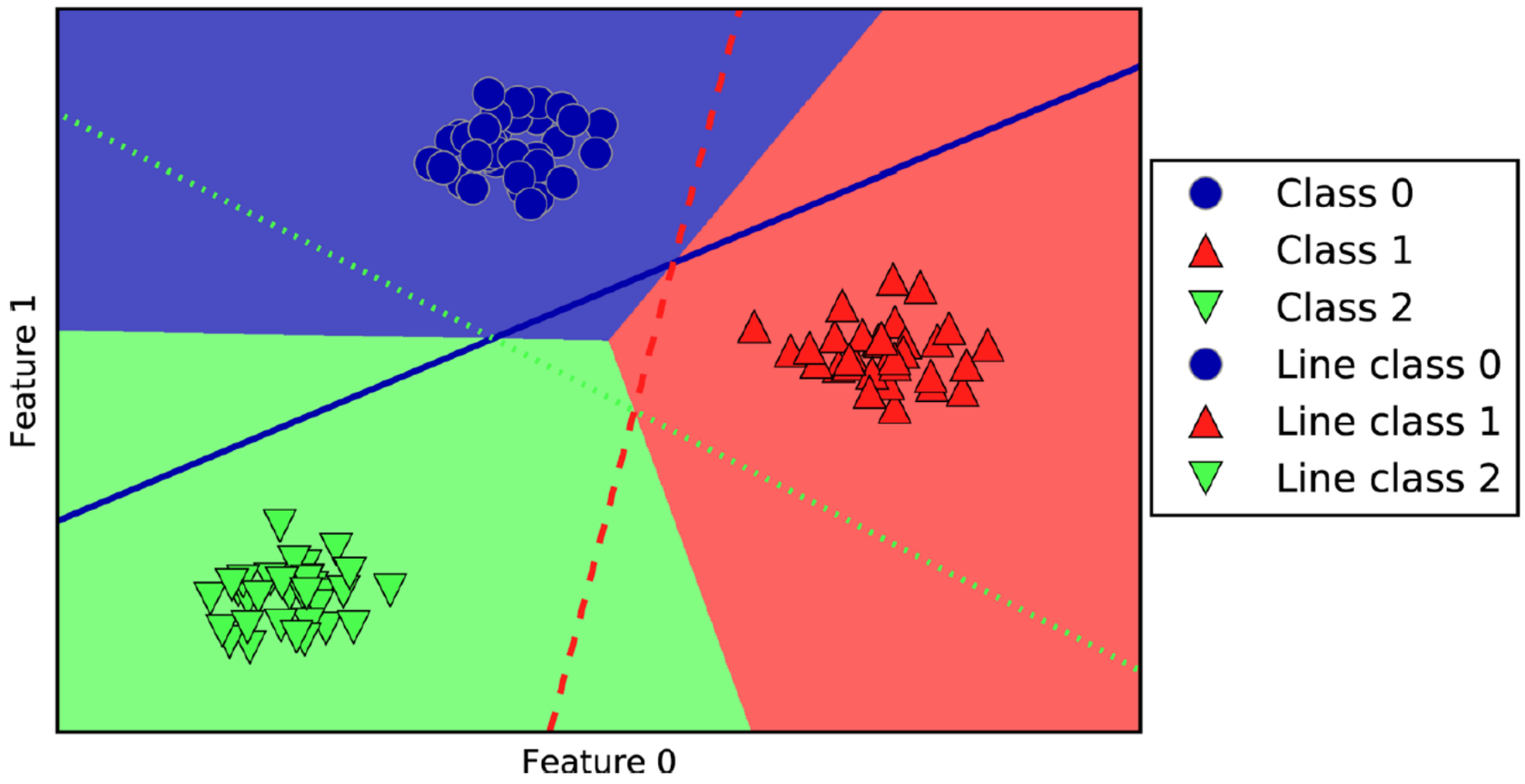


Linear models per la classificazione multiclass

- Il seguente codice mostra le regioni associate alle predizioni

```
mglearn.plots.plot_2d_classification(linear_svm, X, fill=True, alpha=.7)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)
for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_,
                                  ['b', 'r', 'g']):
    plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
plt.legend(['Class 0', 'Class 1', 'Class 2', 'Line class 0', 'Line class 1',
            'Line class 2'], loc=(1.01, 0.3))
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

Linear models per la classificazione multiclass



Considerazioni sul tuning

- Gli iperparametri C e λ sono solitamente campionati su scala logaritmica.
- Se si assume che il dataset contenga solo alcune feature rilevanti si può testare la L1 regularization, altrimenti si tende a preferire la L2.
- La L1 regularization è utile anche per dare una interpretazione del modello.
- I modelli lineari sono veloci da addestrare, anche su dati sparsi.
- Per dataset con >100.000 istanze si può impiegare il parametro *solver='sag'* nella LogisticRegression e Ridge, che rende l'apprendimento più veloce. Altre opzioni sono SGDClassifier e SGDRegressor che implementano versioni più scalabili dei relativi algoritmi.
- I parametri possono indicare quali feature siano più rilevanti, nei casi in cui le feature sono indipendenti tra loro.
- I modelli lineari hanno buone performance quando il numero di features è grande rispetto al numero di istanze. Sono impiegati anche su grandi dataset perché spesso gli altri modelli non sono facilmente addestrabili.

Testi di Riferimento

- Andreas C. Müller, Sarah Guido. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media 2016
- Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media 2017