Aprendizaje supervisado

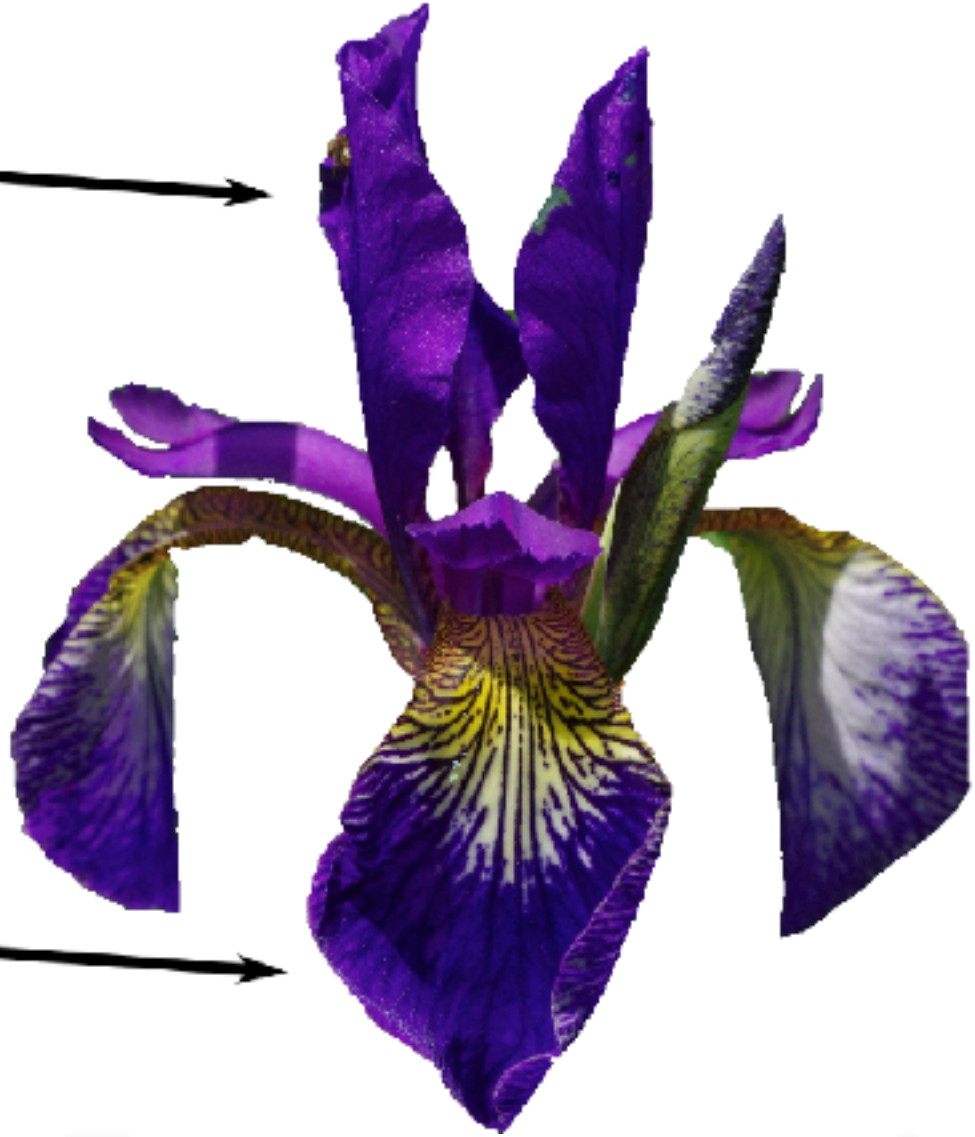
Aprendizaje supervisado

- Entregar entradas y salidas deseadas.
- Máquina produce algoritmo
- Predicciones para nuevas entradas.
- Ejemplos
  - Reconocer escritura a mano
  - Diagnóstico de tumores
  - Actividad fraudulenta en tarjetas de crédito

Clasificación y Regresión

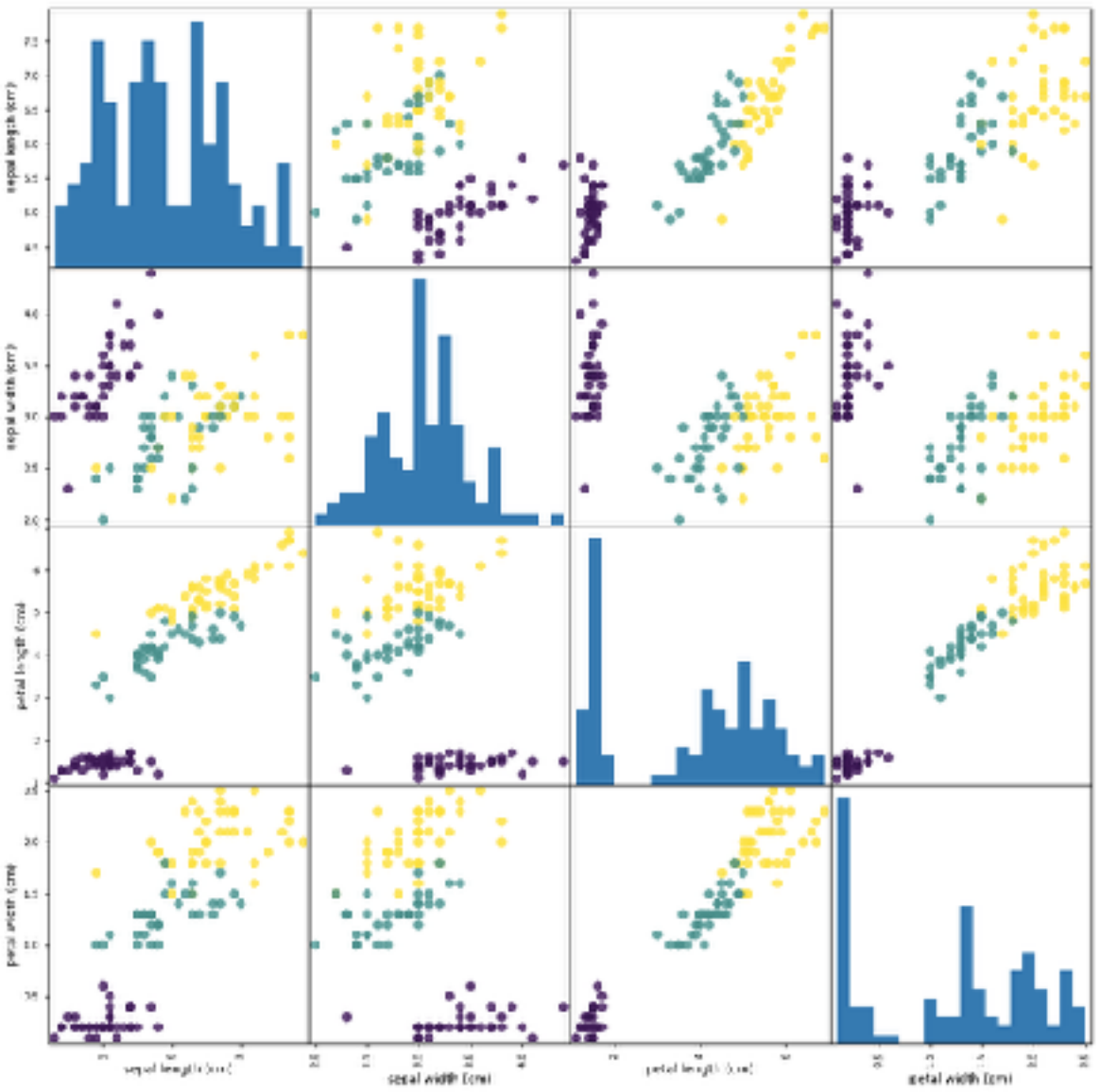
- Clasificación
  - Predecir *etiquetas* de una lista predefinida de posibilidades.
  - Clasificación binaria
  - Clasificación multiclase
- Regresión
  - Predecir de una forma continua un número real

Petal



Sepal

Iris Data Set



K-Neighbors

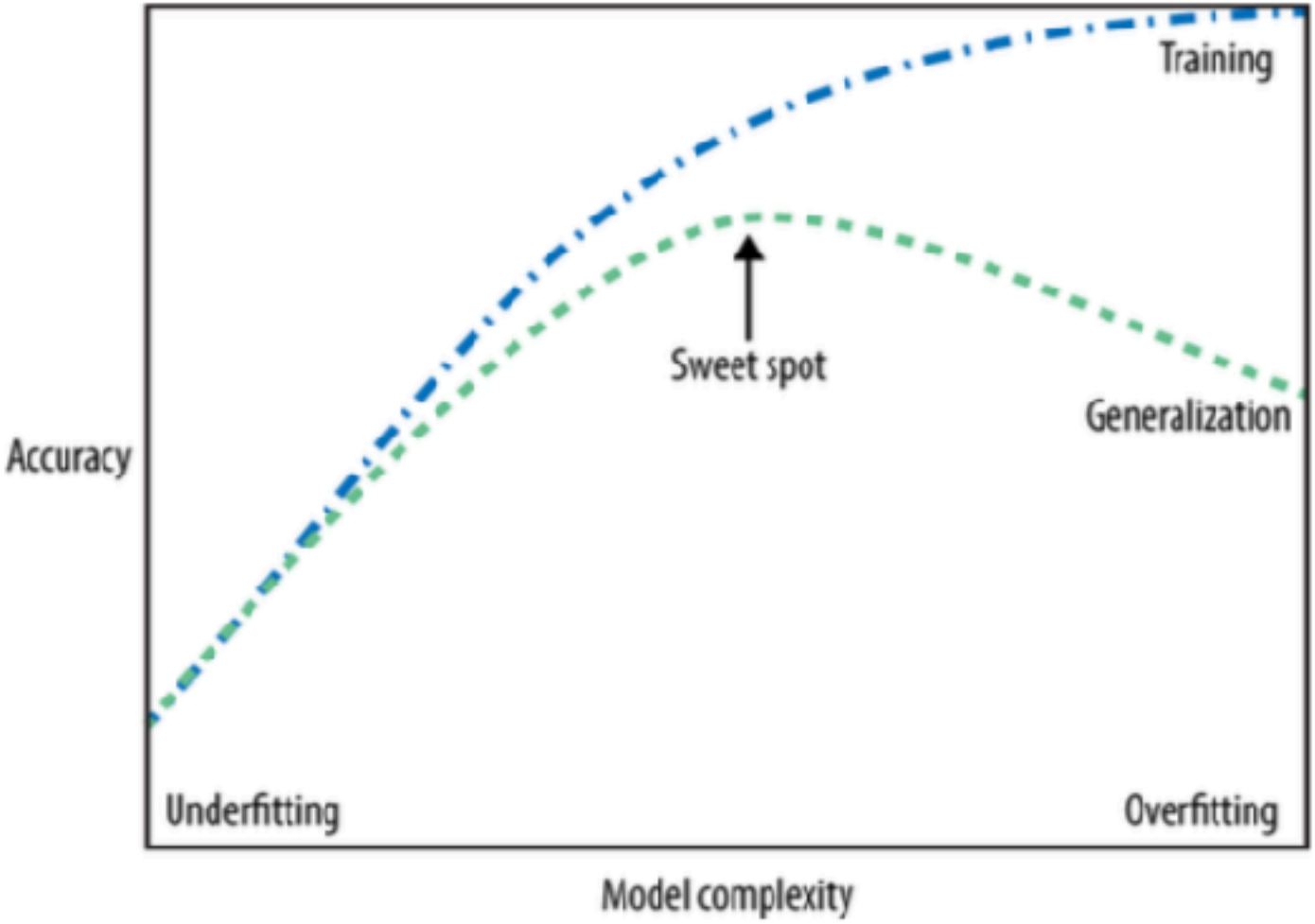
```
from sklearn.datasets import load_iris
iris_dataset = load_iris()


X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)

knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(X_train, y_train)

print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

```
Test set score: 0.97
```

Reglas

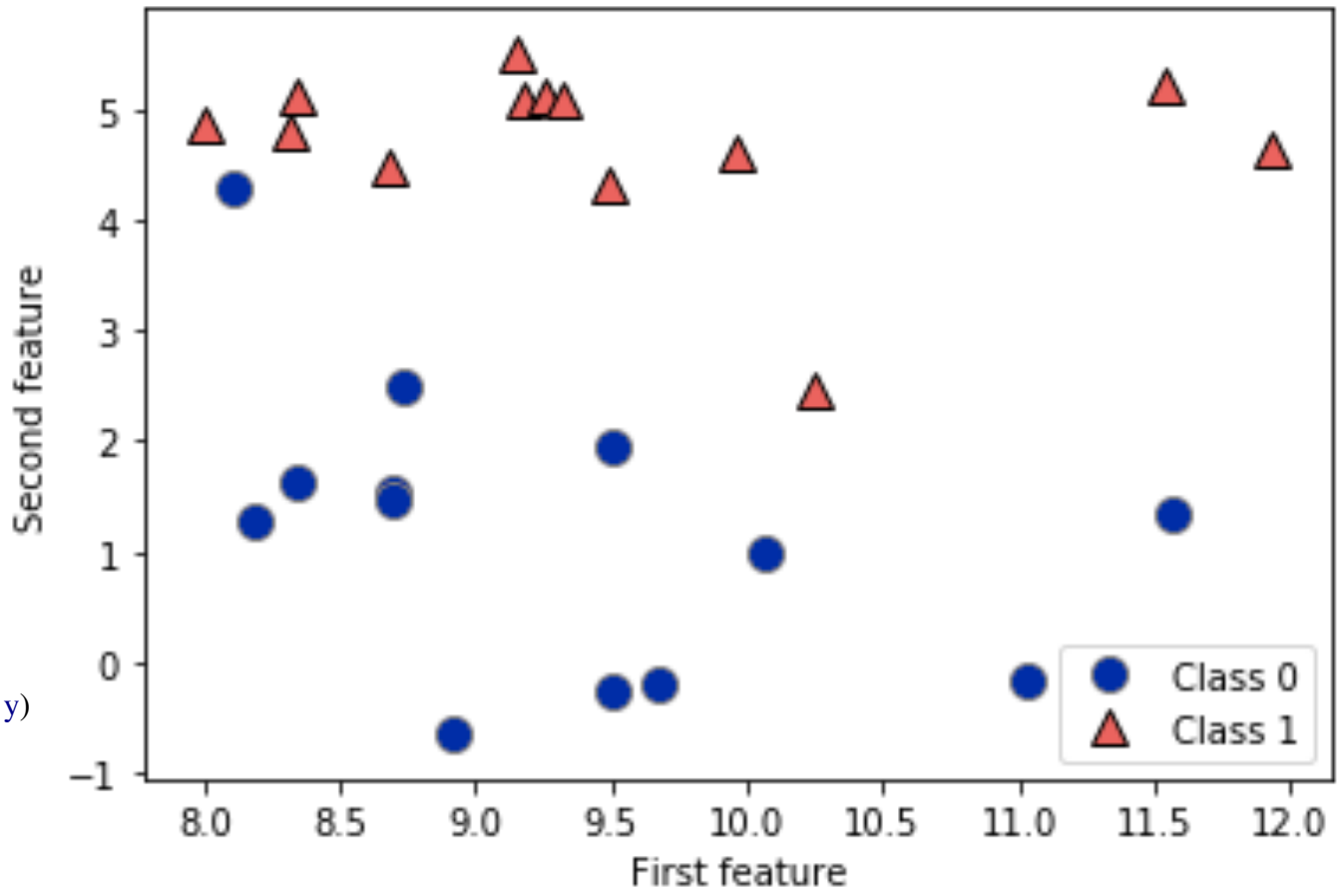
Age	Number of cars owned	Owns house	Number of children	Marital status	Owns a dog	Bought a boat
66	1	yes	2	widowed	no	yes
52	2	yes	3	married	no	yes
22	0	no	0	married	yes	no
25	1	no	1	single	no	no
44	0	no	2	divorced	yes	no
39	1	yes	2	married	yes	no
26	1	no	2	single	no	no
40	3	yes	1	married	yes	no
53	2	yes	2	divorced	no	yes
64	2	yes	3	divorced	no	no
58	2	yes	2	married	yes	yes
33	1	no	1	single	no	no

Fitting

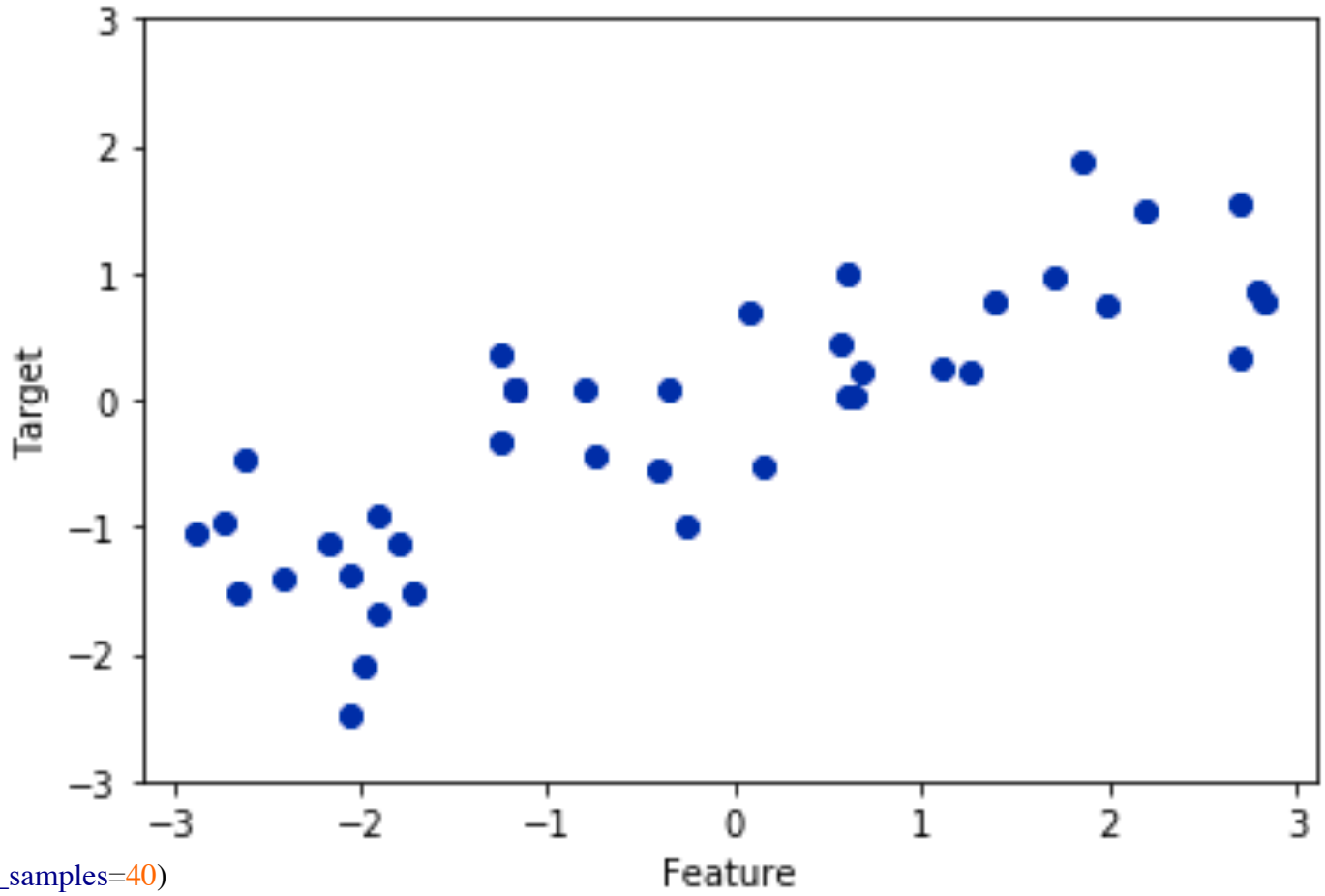

Training
Sweet spot
Generalization
Accuracy
Underfitting
Overfitting
Model complexity

Clasificación

```
X, y = mglearn.datasets.make_forge()
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.legend(["Class 0", "Class 1"], loc=4)
plt.xlabel("First feature")
plt.ylabel("Second feature")
```

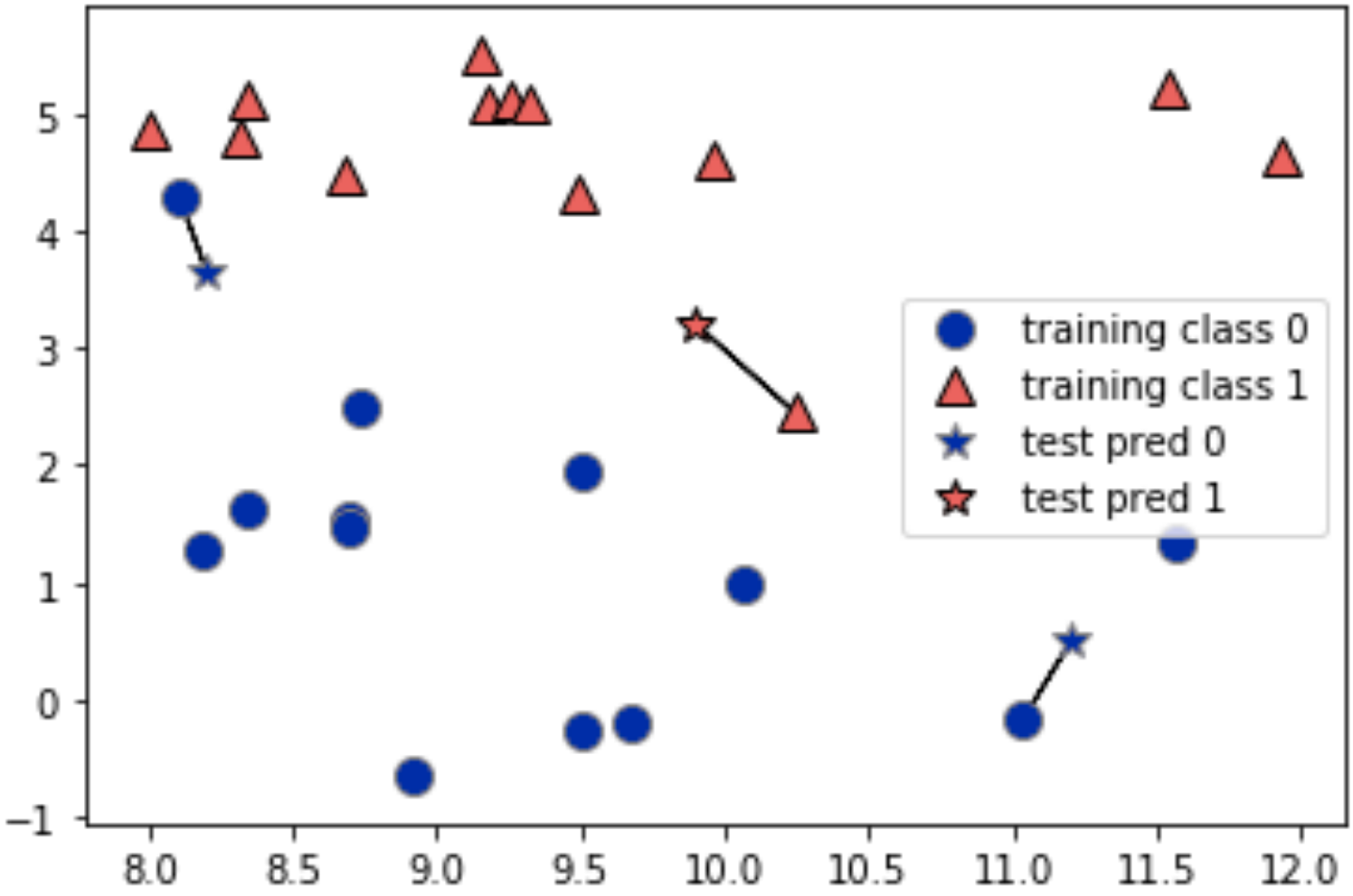


Objetivos



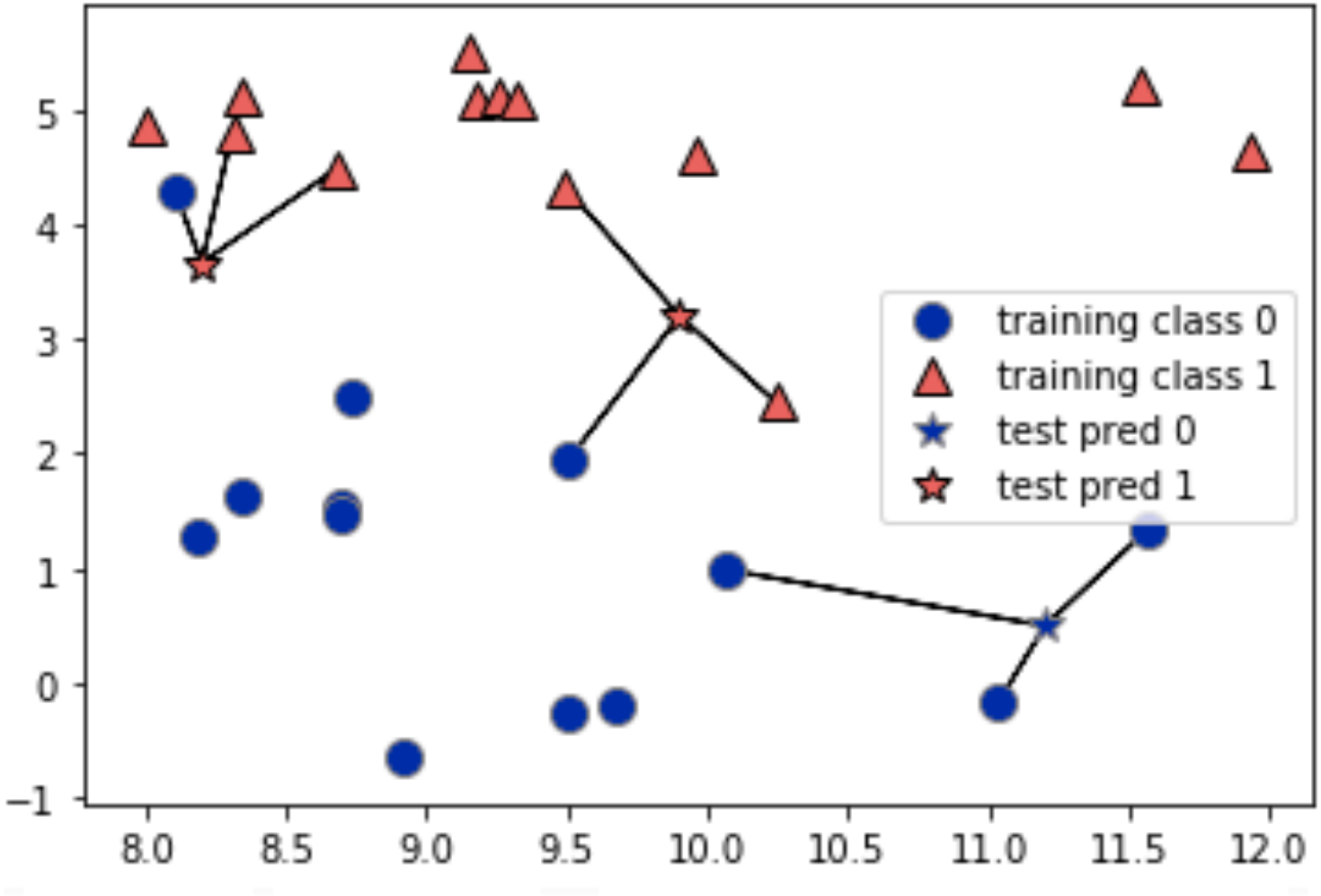

```
X, y = mglearn.datasets.make_wave(n_samples=40)
  plt.plot(X, y, 'o')
  plt.ylim(-3, 3)
  plt.xlabel("Feature")
  plt.ylabel("Target")
```

k-Neighbors




```
mglearn.plots.plot_knn_classification(n_neighbors=1)
```

K-Neighbors


training class 0
training class 1
test pred 0
test pred 1
5
4
3
2
1
0
−1
8.0
8.5
9.0
9.5
10.0
10.5
11.0
11.5
12.0

K- Neighbors

```
mglearn.plots.plot_knn_classification(n_neighbors=3)
```

```
from sklearn.model_selection import train_test_split
X, y = mglearn.datasets.make_forge()

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)

clf.fit(X_train, y_train)
```

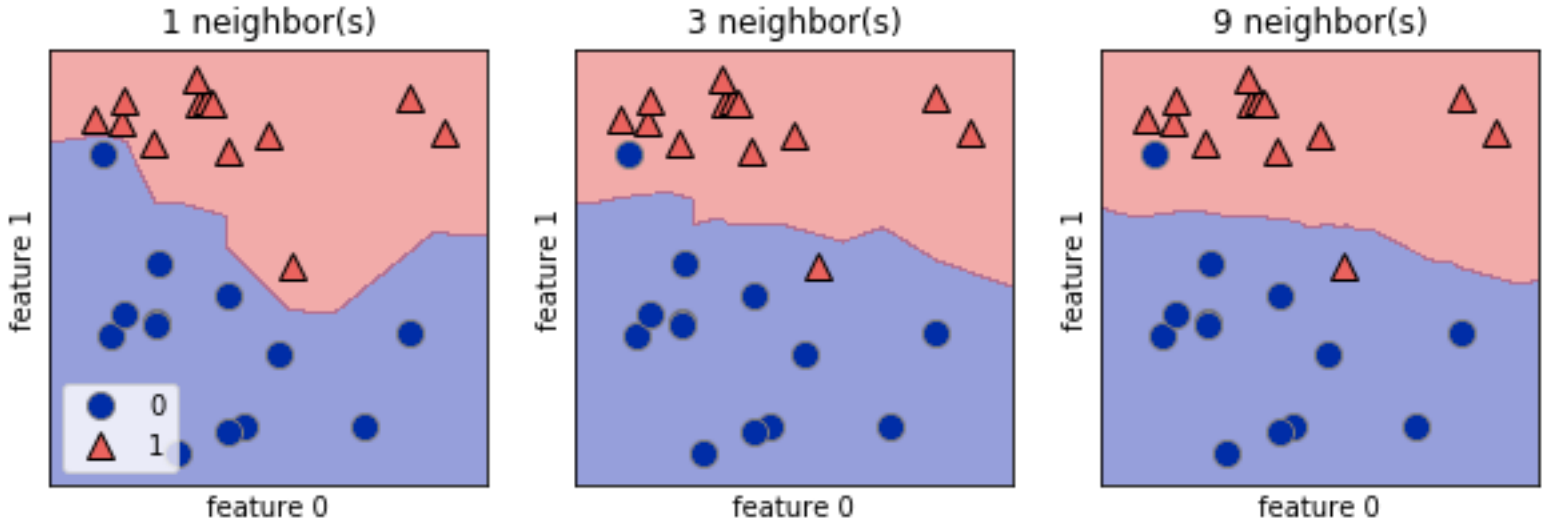
```
print("Test set predictions: {}".format(clf.predict(X_test)))
```

```
Test set predictions: [1 0 1 0 1 0 0]
```

```
print("Test set accuracy: {:.2f}".format(clf.score(X_test, y_test)))
```

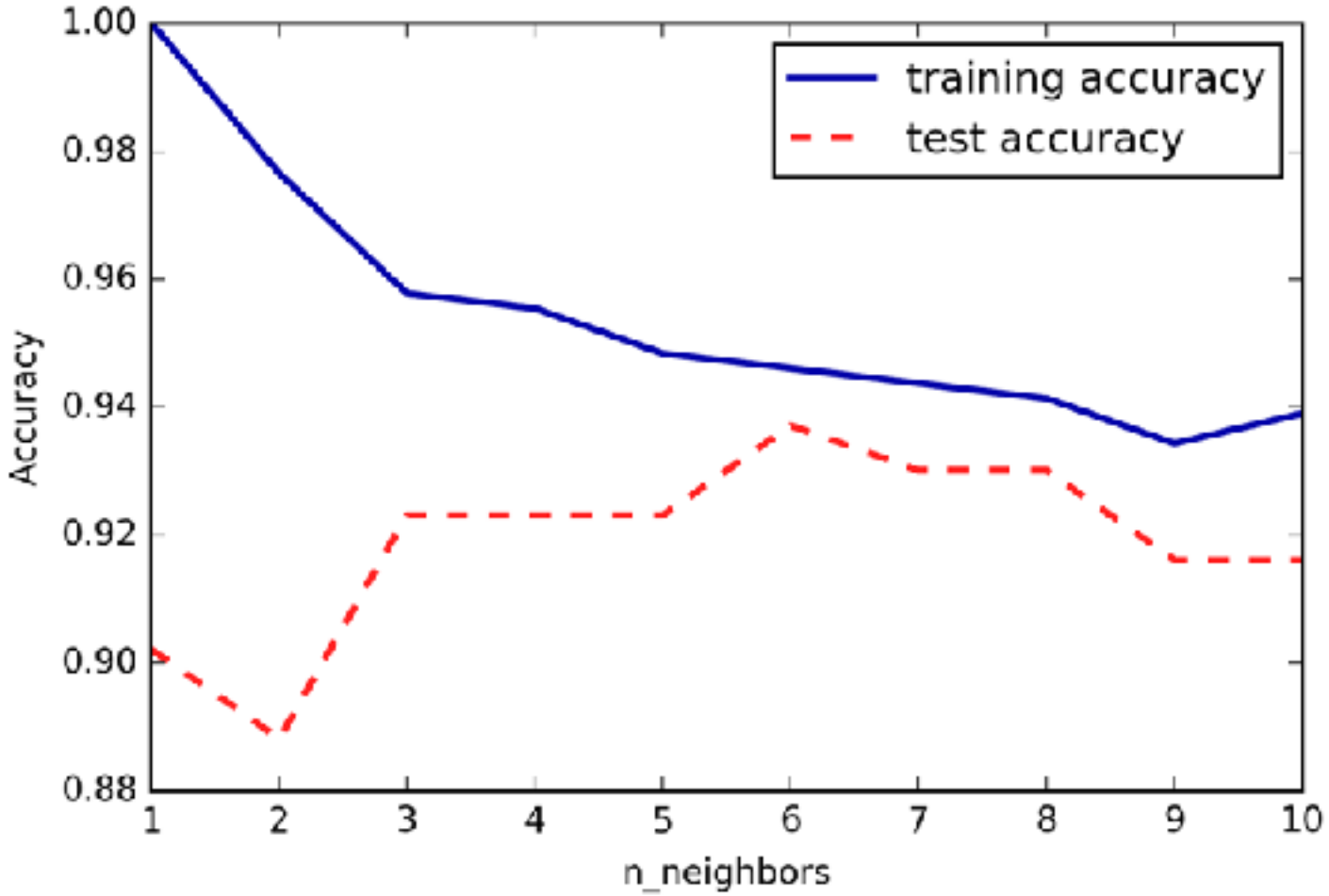
```
Test set accuracy: 0.86
```

Análisis de k-neighbors

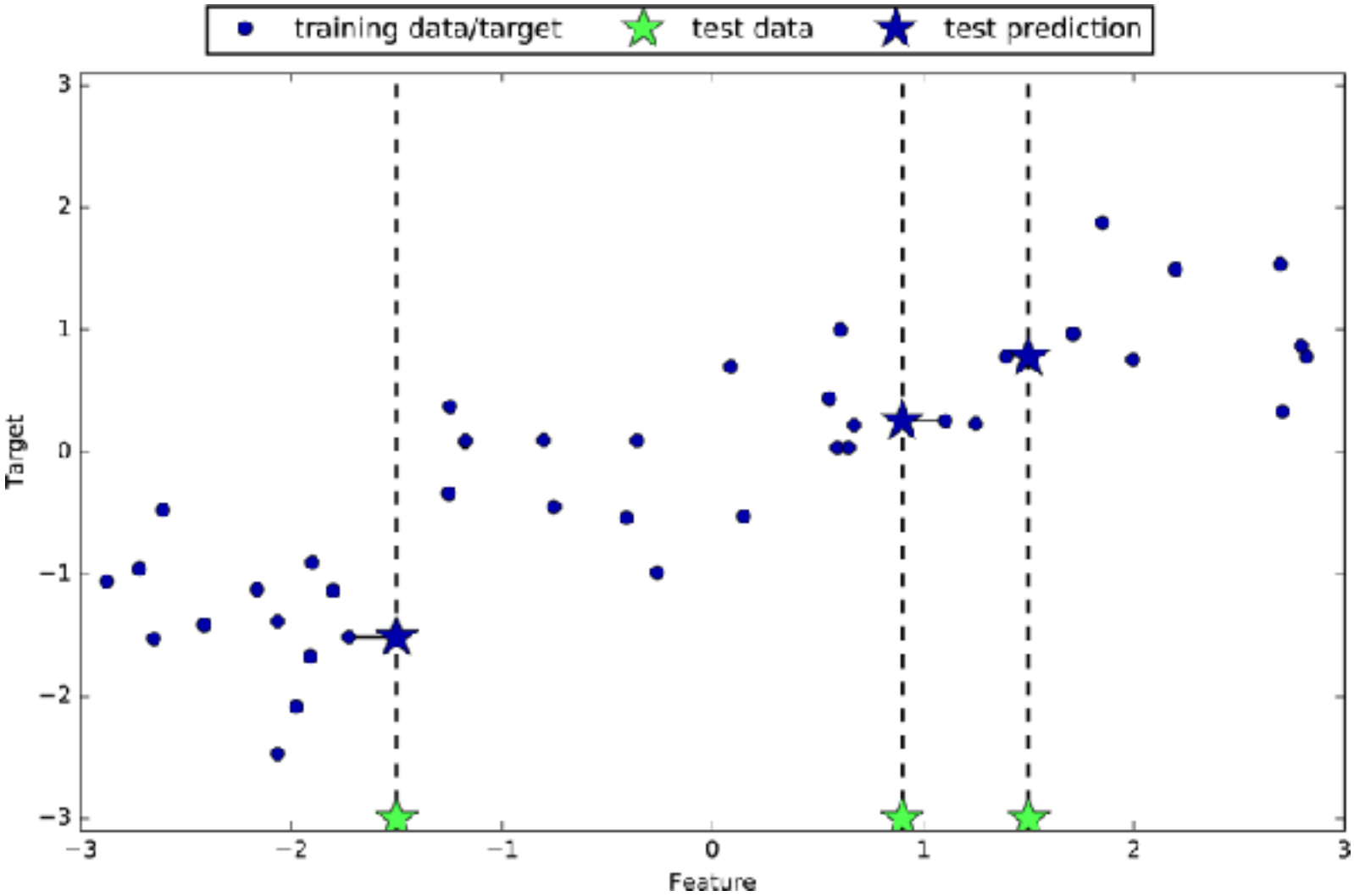



```
 fig, axes = plt.subplots(1, 3, figsize=(10, 3))
for n_neighbors, ax in zip([1, 3, 9], axes):
# the fit method returns the object self, so we can instantiate
# and fit in one line
clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X, y) mglearn.plots.plot_2d_separator(clf, X, fill=True, eps=0.5, ax=ax, alpha=.4
mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
ax.set_title("{} neighbor(s)".format(n_neighbors))
ax.set_xlabel("feature 0")
ax.set_ylabel("feature 1")

    axes[0].legend(loc=3)
```

Precisión entrenamiento y prueba

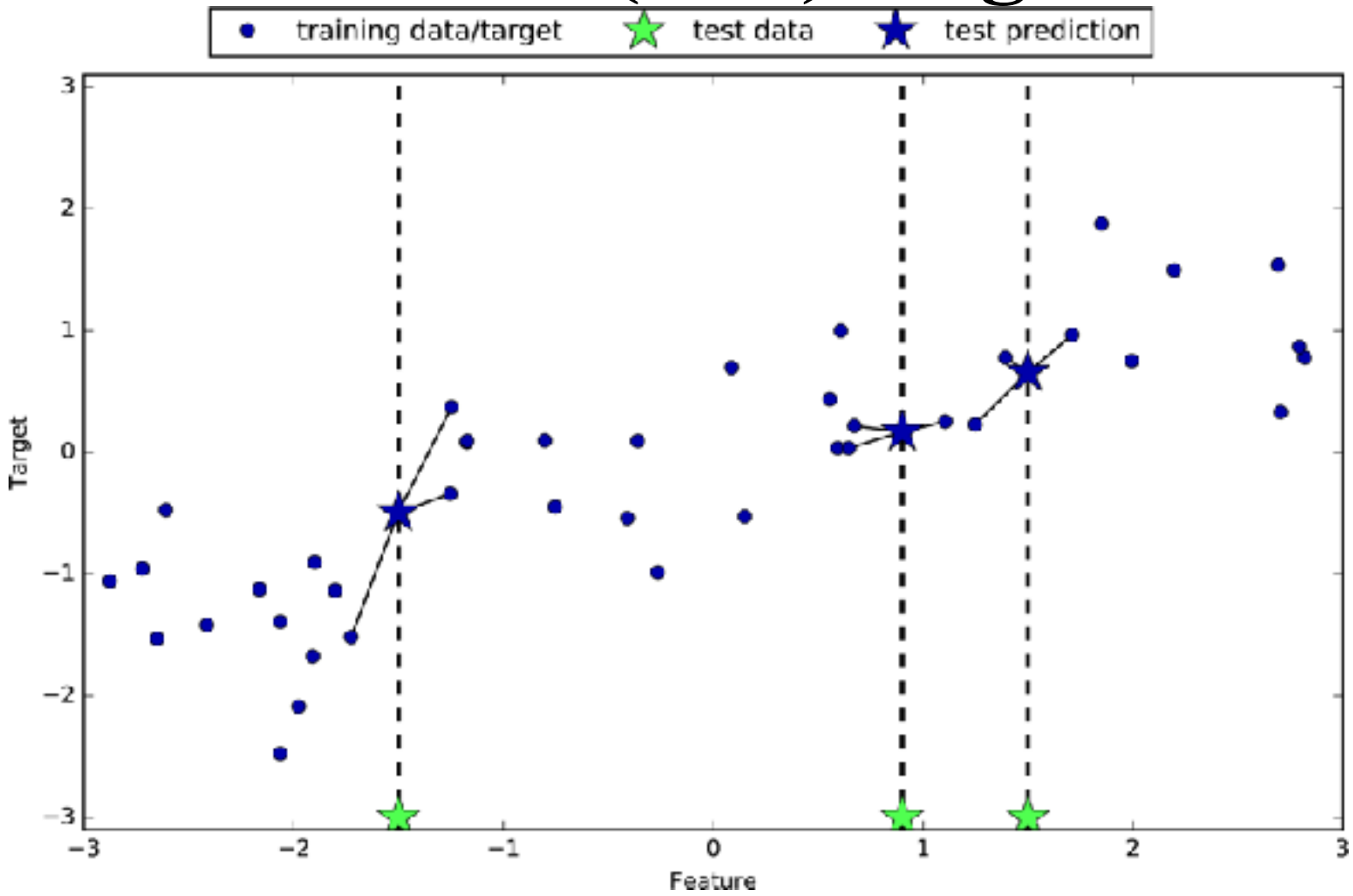


k-vecinos (k=1) Regresión



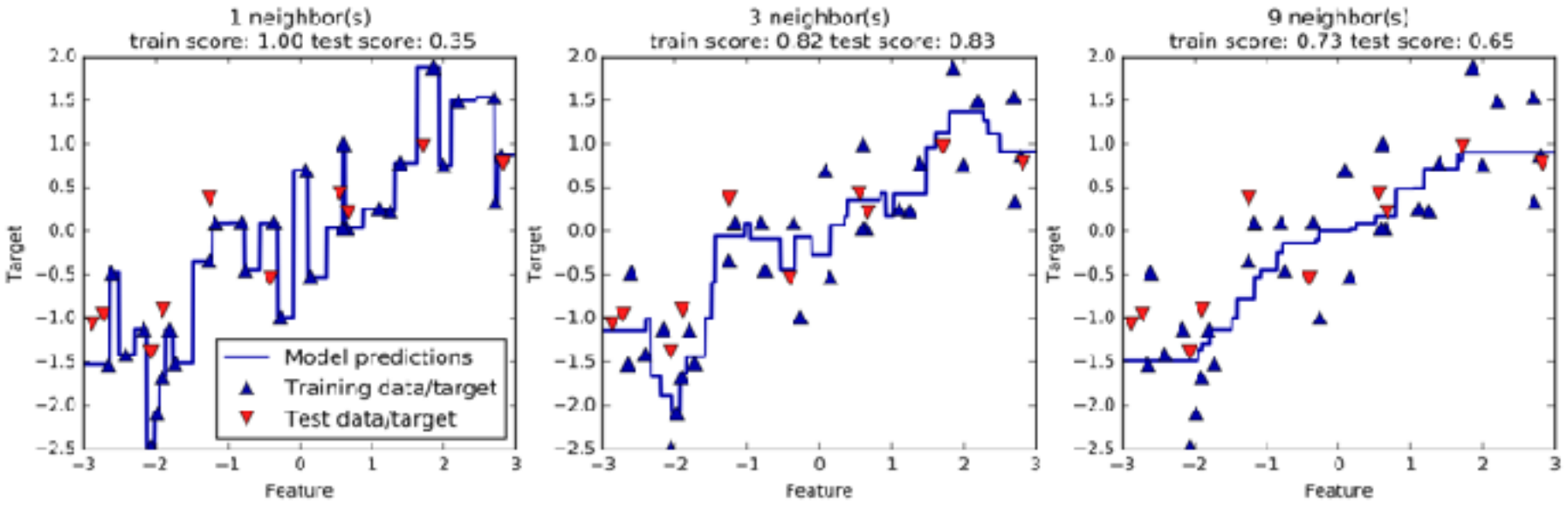
mglearn.plots.plot_knn_regression(n_neighbors=3)

k-vecinos (k=3) Regresión



mglearn.plots.plot_knn_regression(n_neighbors=3)

Comparación regresiones



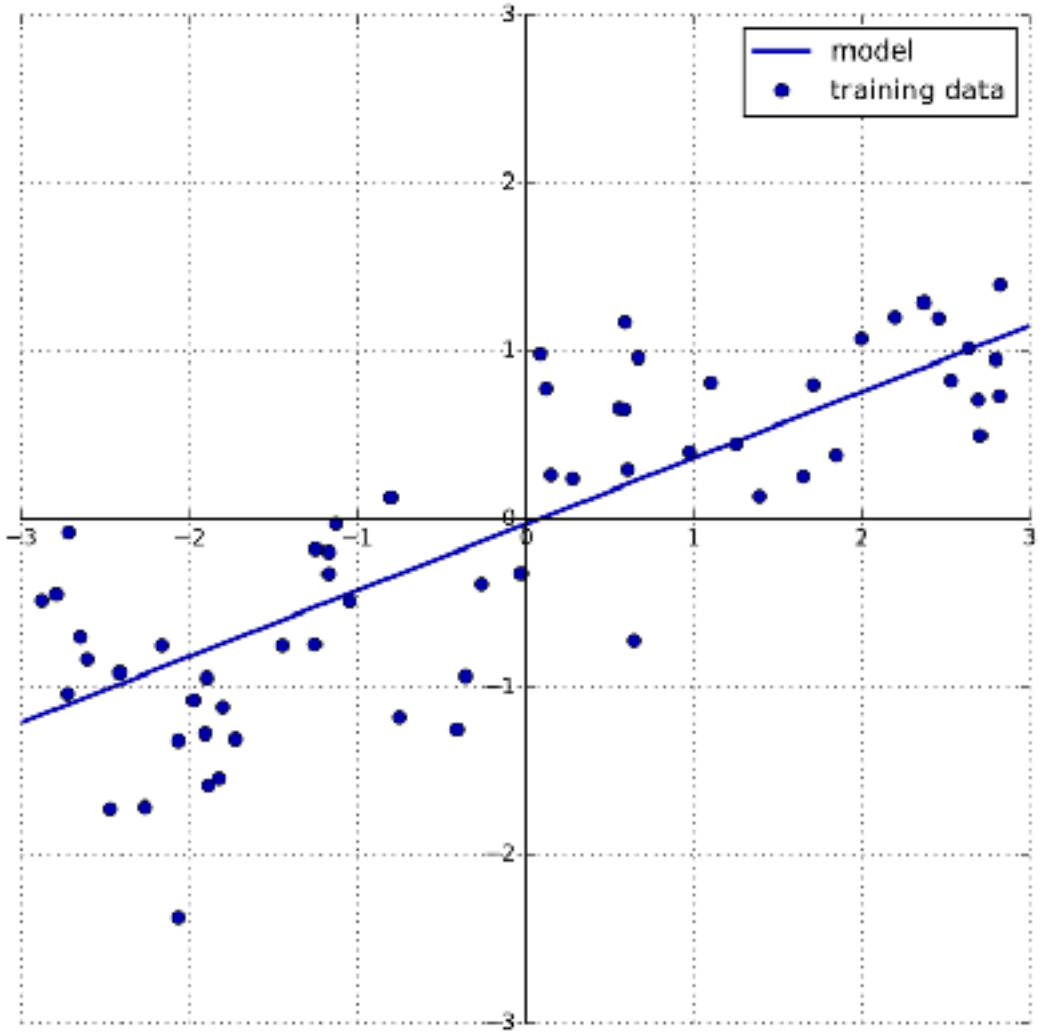
Modelos lineales

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \ldots + w[p] * x[p] + b$$

Ejemplo de modelo lineal

$$\hat{y} = w[0] * x[0] + b$$

w[0]: 0.393906 b: -0.031804



Mínimos cuadrados

```
from sklearn.linear_model import LinearRegression
X, y = mglearn.datasets.make_wave(n_samples=60)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

lr = LinearRegression().fit(X_train, y_train)
        print("lr.coef_: {}".format(lr.coef_))

        print("lr.intercept_: {}".format(lr.intercept_))
```

```
lr.coef_: [ 0.394]
lr.intercept_: -0.031804343026759746
```

```
print("Training set score: {:.2f}".format(lr.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lr.score(X_test, y_test)))
```

```
Training set score: 0.67
  Test set score: 0.66
```

Regresión ridge

- Modelo lineal de regresión
- Coeficientes tan pequeños como sea posible
- Regularización para evitar *overfitting*

```
from sklearn.linear_model import Ridge

ridge = Ridge().fit(X_train, y_train)
print("Training set score: {:.2f}".format(ridge.score(X_train, y_train)))

print("Test set score: {:.2f}".format(ridge.score(X_test, y_test)))
```

```
Training set score: 0.89
Test set score: 0.75
```

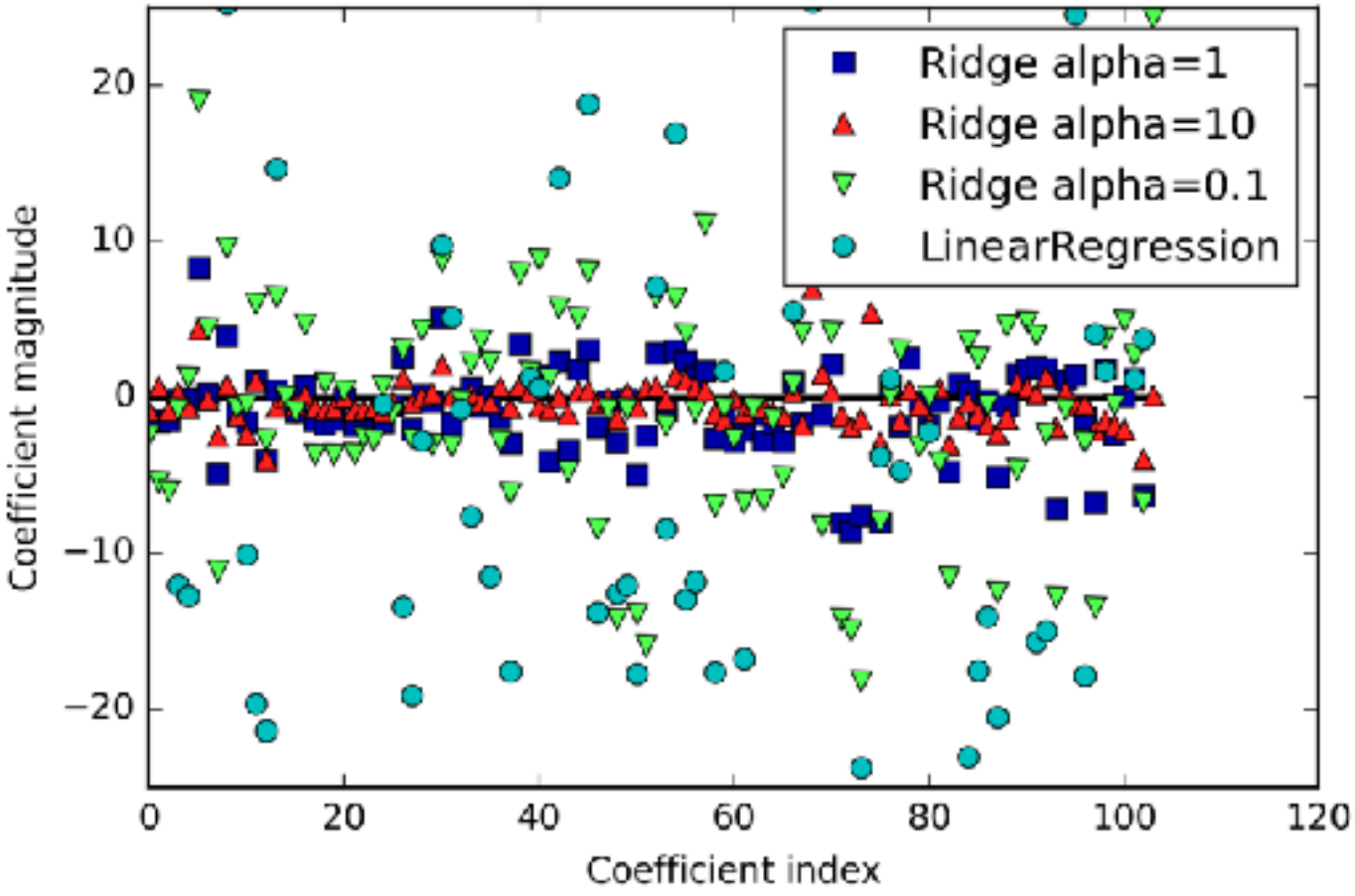
```
ridge10 = Ridge(alpha=10).fit(X_train, y_train)
print("Training set score: {:.2f}".format(ridge10.score(X_train, y_t

print("Test set score: {:.2f}".format(ridge10.score(X_test, y_test)))
```

```
Training set score: 0.79
Test set score: 0.64
```

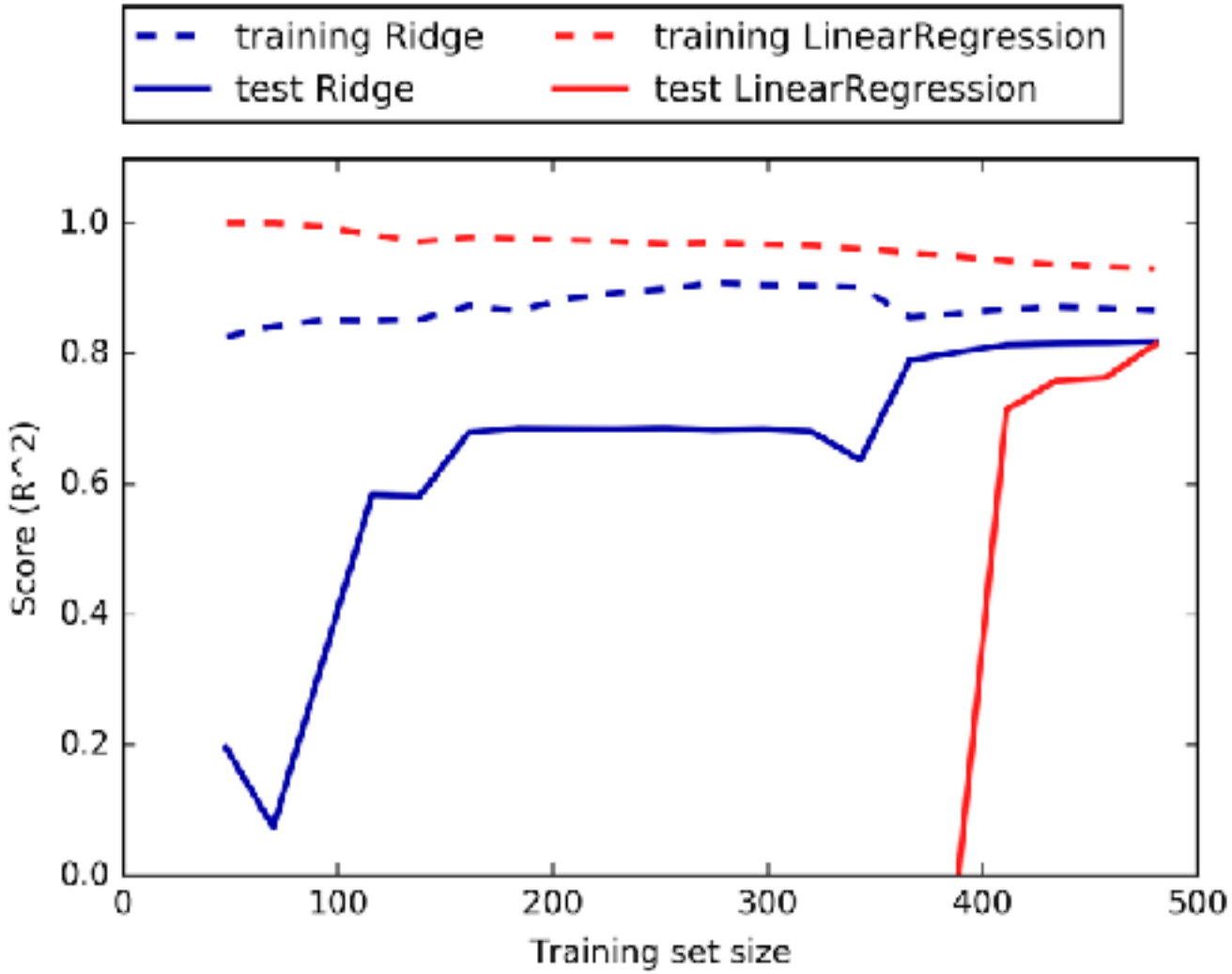
```
ridge01 = Ridge(alpha=0.1).fit(X_train, y_train)
print("Training set score: {:.2f}".format(ridge01.score(X_train, y_train)))

print("Test set score: {:.2f}".format(ridge01.score(X_test, y_test)))
```

```
Training set score: 0.93
Test set score: 0.77
```

Comparación regresión ridge



Curvas de aprendizaje



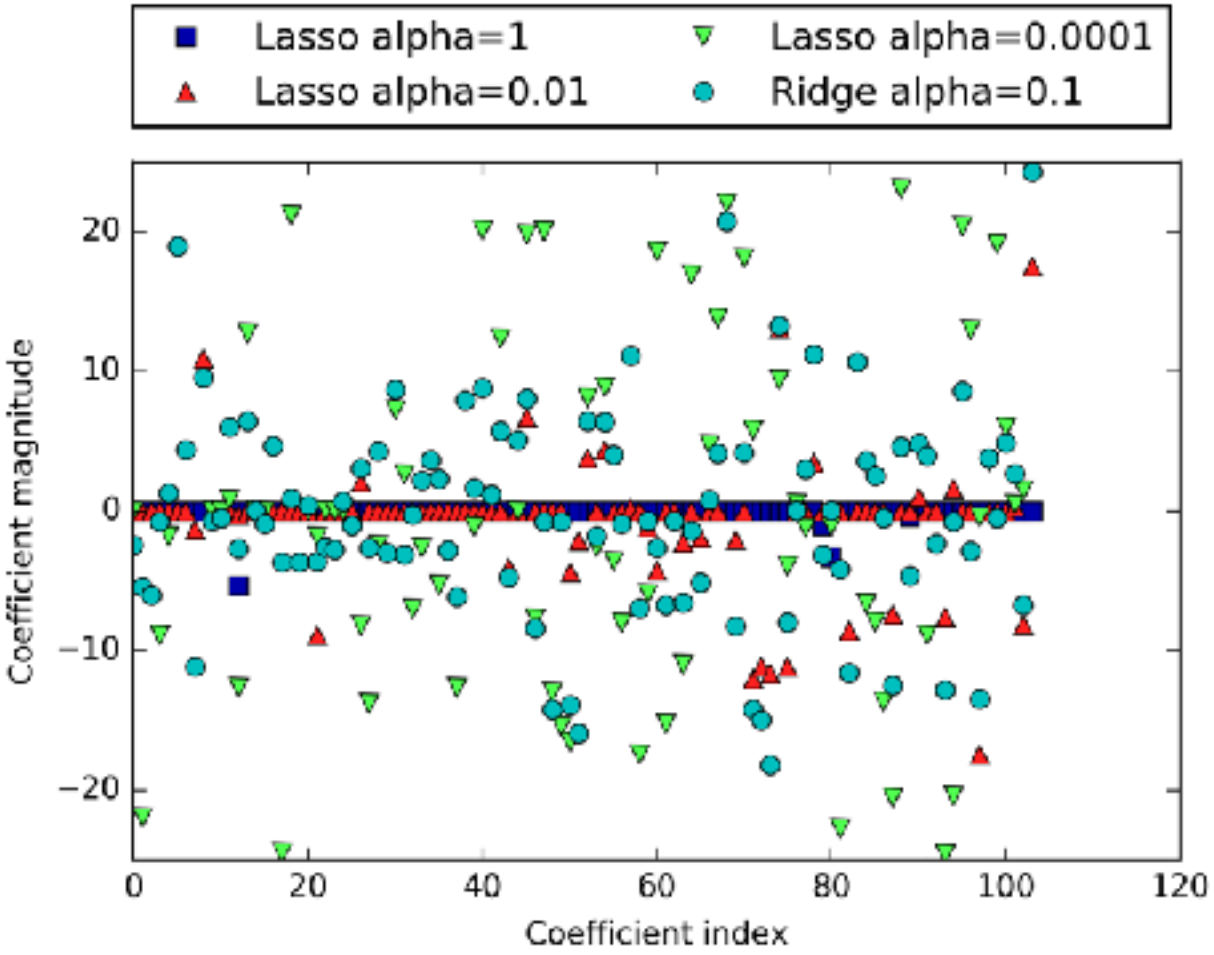
Lasso

- Regularización L1 - algunos coeficientes = 0

```
from sklearn.linear_model import Lasso

lasso = Lasso().fit(X_train, y_train)
print("Training set score: {:.2f}".format(lasso.score(X_train, y_train))) print("Test set score: {:.2f}".format(lasso.score(X_test, y_test)))
print("Number of features used: {}".format(np.sum(lasso.coef_ != 0)))
```

```
Training set score: 0.29
  Test set score: 0.21
  Number of features used: 4
```

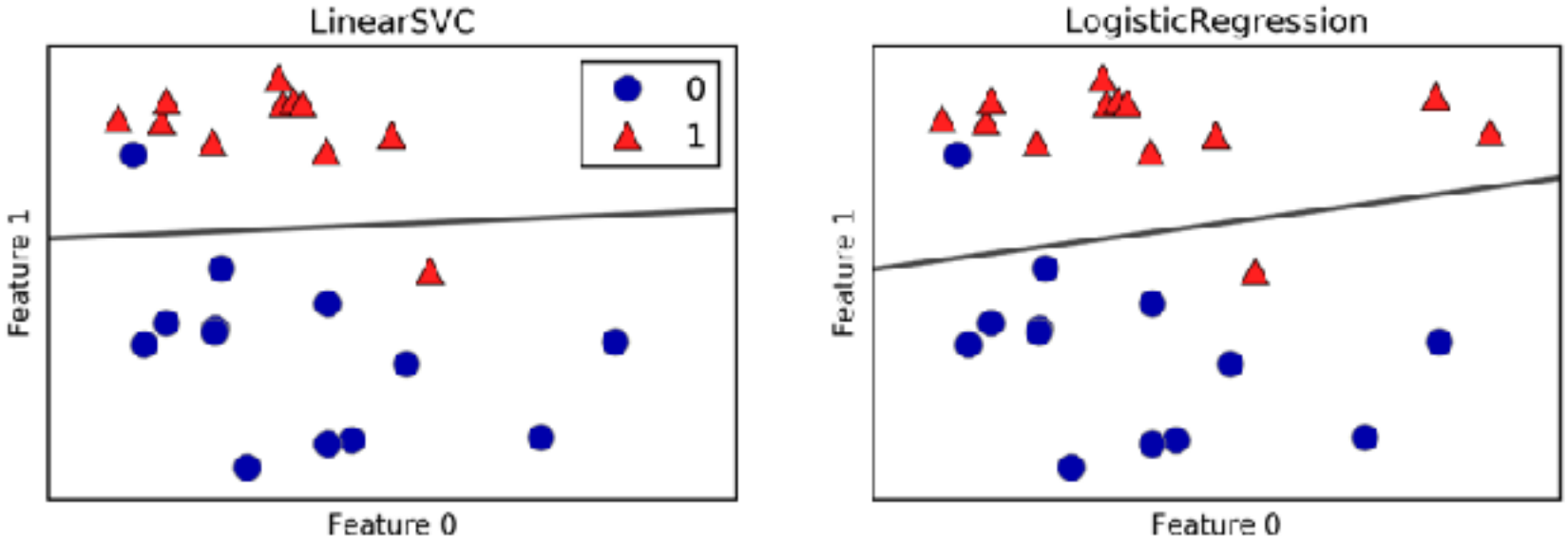
Lasso comparado con Ridge



Clasificación lineal

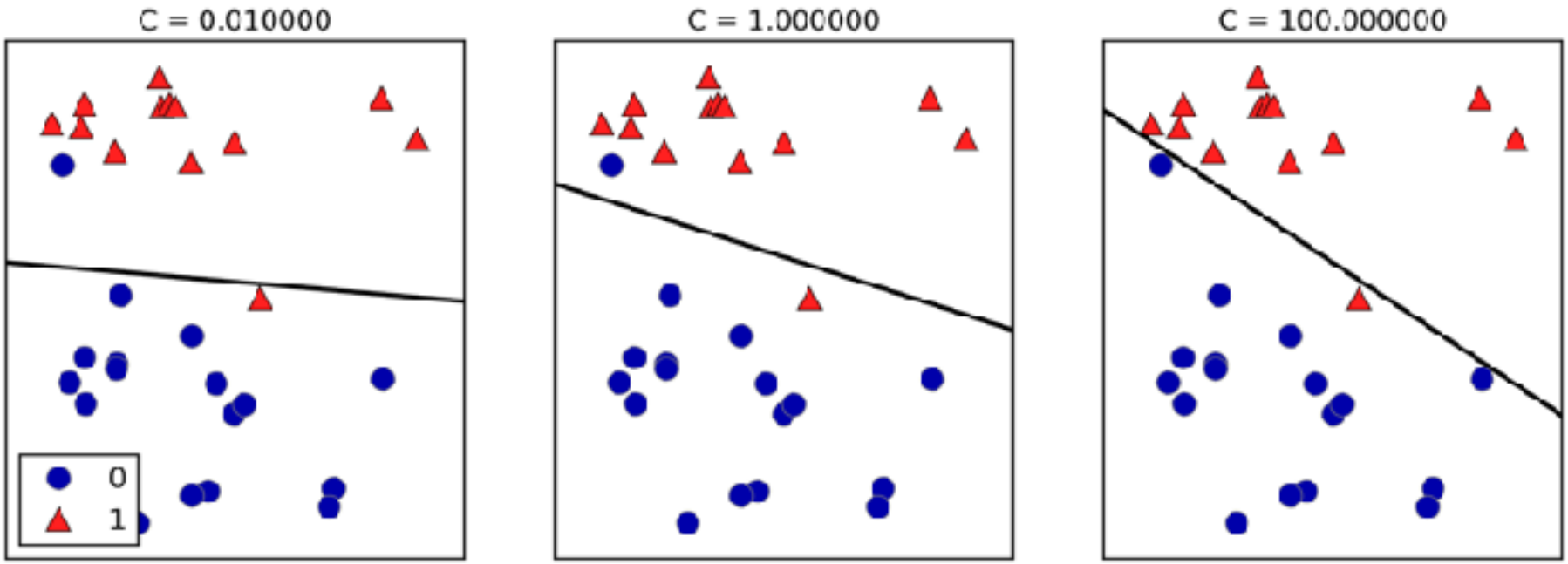
$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \ldots + w[p] * x[p] + b > 0$$

Vector de soporte (SVC) y Regresión logística




```
    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()
```

Sensibilidad a C - SVC



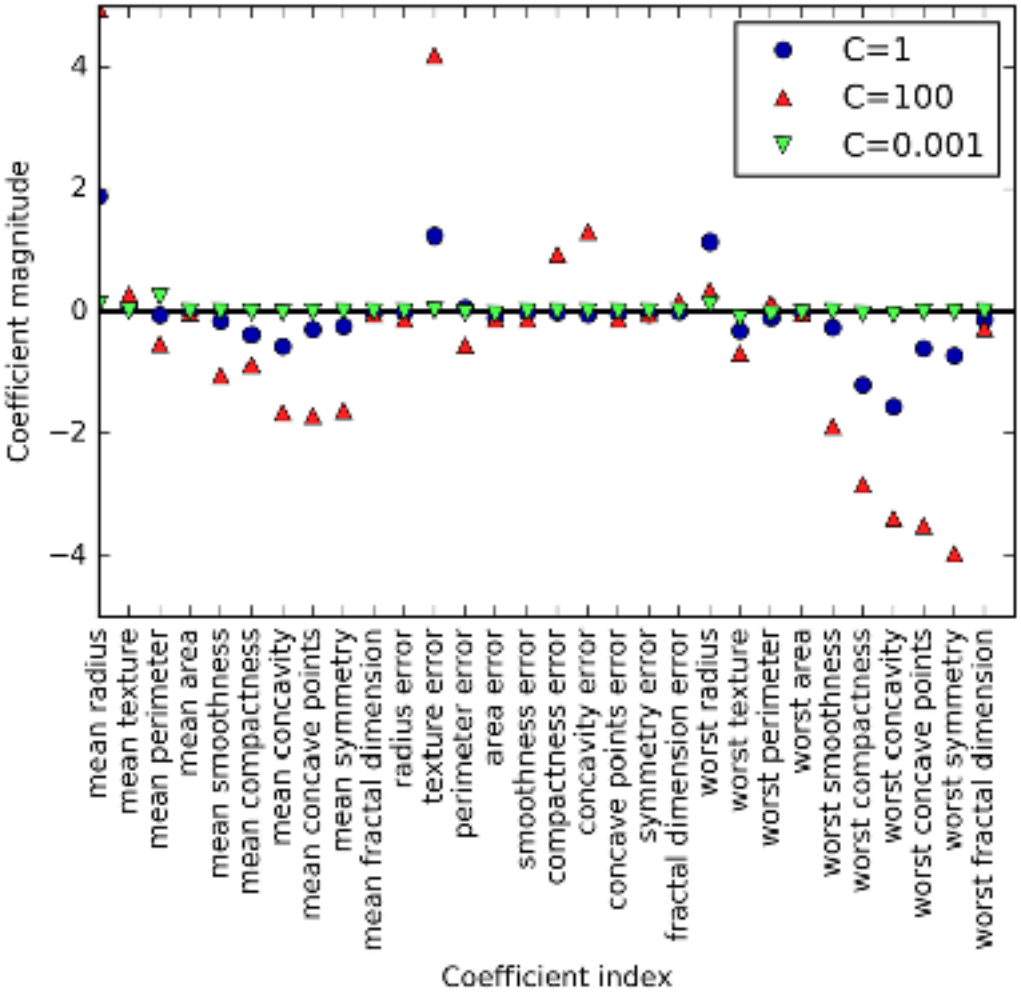
mglearn.plots.plot_linear_svc_regularization()

Sensibilidad a C - Regresión logística

```
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().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
logreg100 = LogisticRegression(C=100).fit(X_train, y_train)
print("Training set score: {:.3f}".format(logreg100.score(X_train,
y_train))) print("Test set score: {:.
3f}".format(logreg100.score(X_test, y_test)))
```

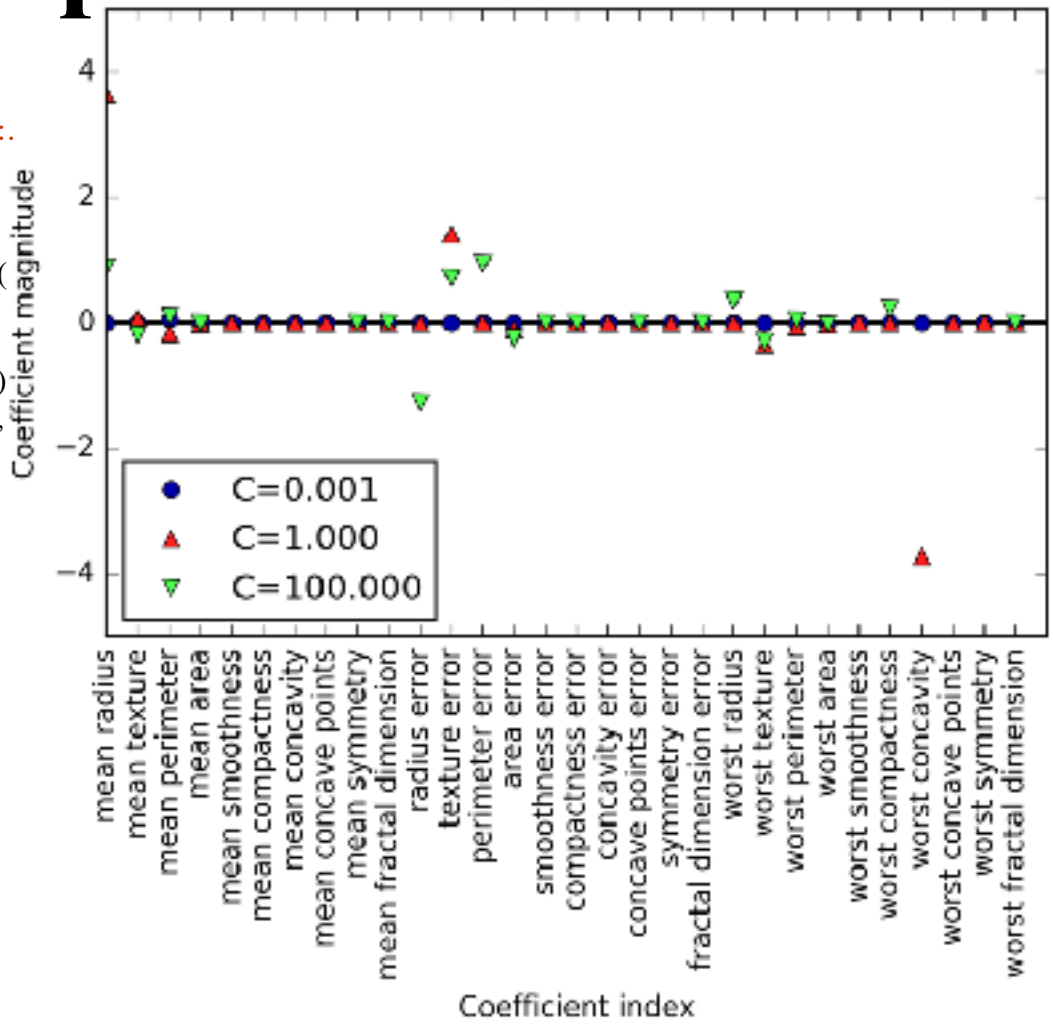
```
 plt.plot(logreg.coef_.T, 'o', label="C=1")
   plt.plot(logreg100.coef_.T, '^', label="C=100")
   plt.plot(logreg001.coef_.T, 'v', label="C=0.001")
   plt.xticks(range(cancer.data.shape[1]), cancer.feature_names,
rotation=90)
   plt.hlines(0, 0, cancer.data.shape[1])
   plt.ylim(-5, 5)
   plt.xlabel("Coefficient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



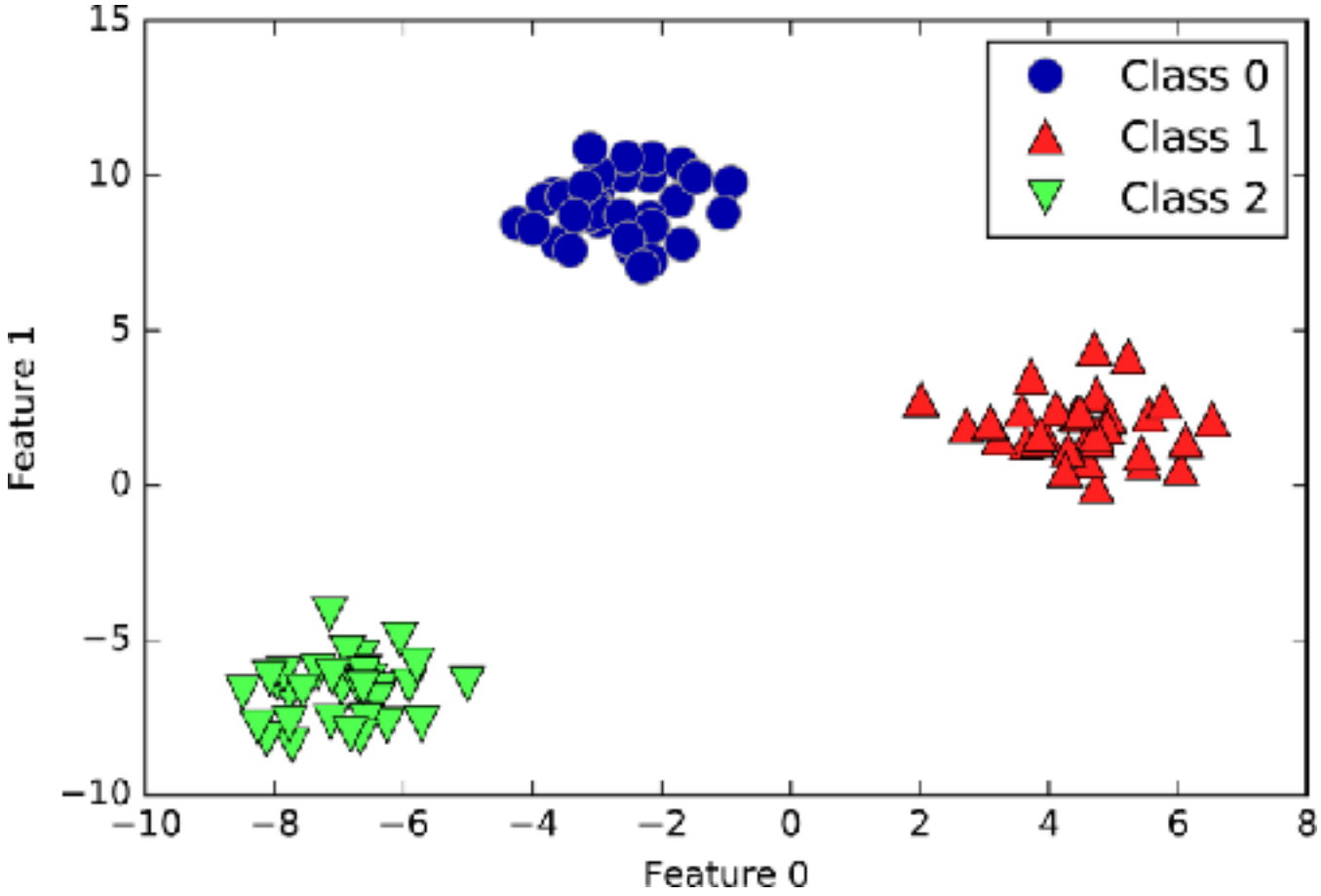
Coeficientes con penalidad

```
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)
```

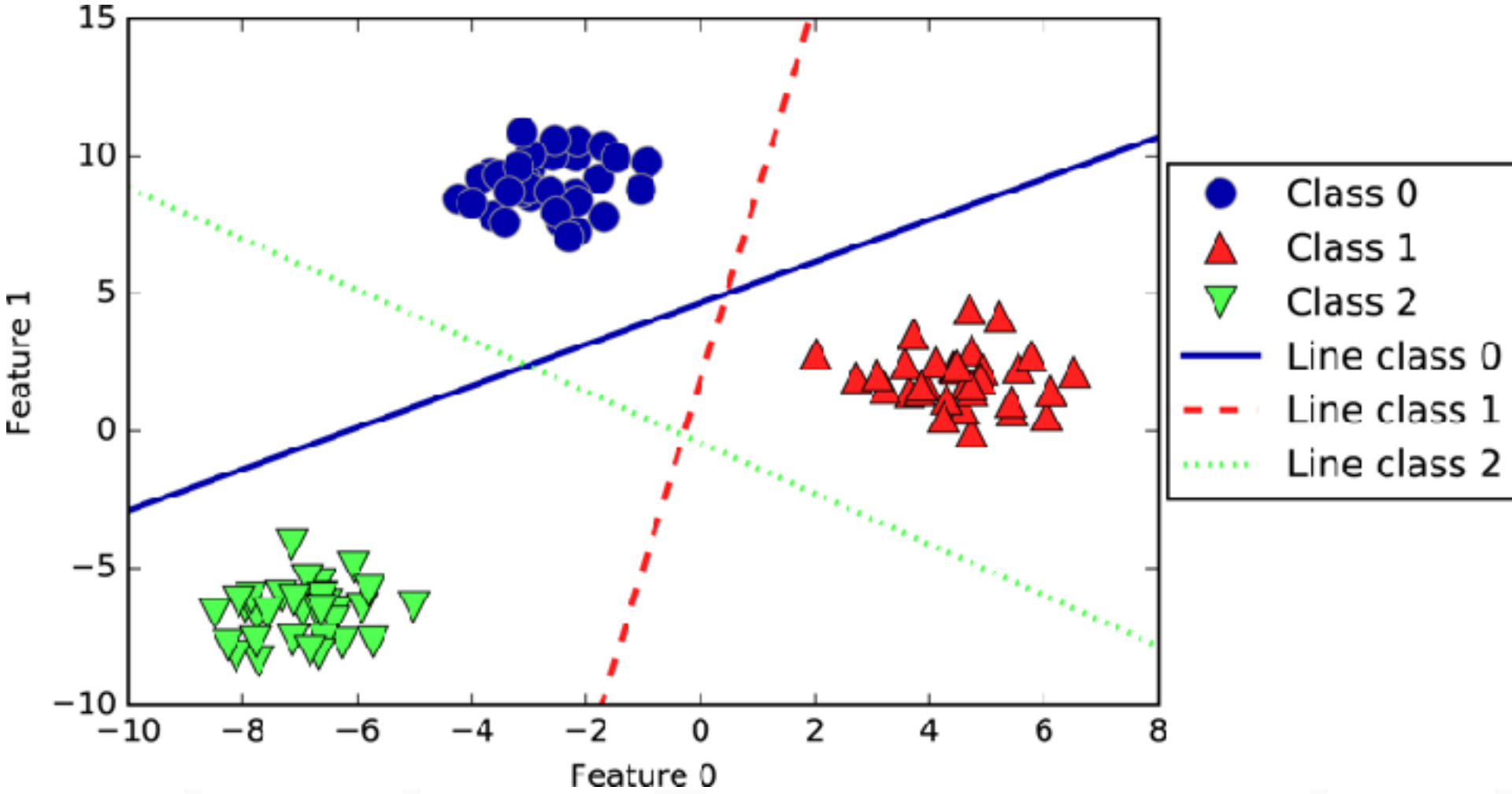
```
Training accuracy of l1 logreg with C=0.001: 0.91
Test accuracy of l1 logreg with C=0.001: 0.92
Training accuracy of l1 logreg with C=1.000: 0.96
Test accuracy of l1 logreg with C=1.000: 0.96
Training accuracy of l1 logreg with C=100.000: 0.99
Test accuracy of l1 logreg with C=100.000: 0.98
```



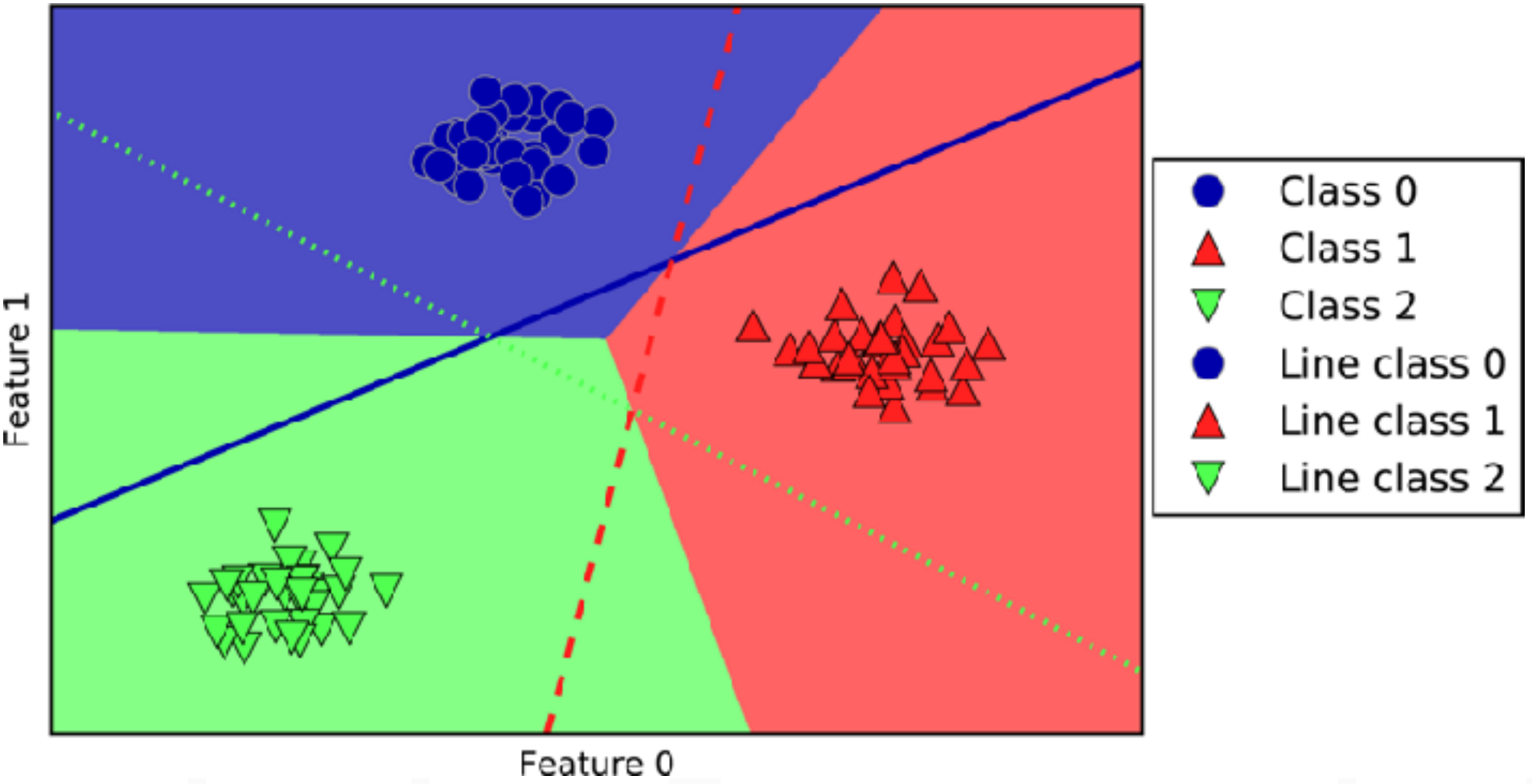
Clasificadores lineales - multiclase



Predicción multiclase - SVC



Decisiones multiclase



Conclusiones

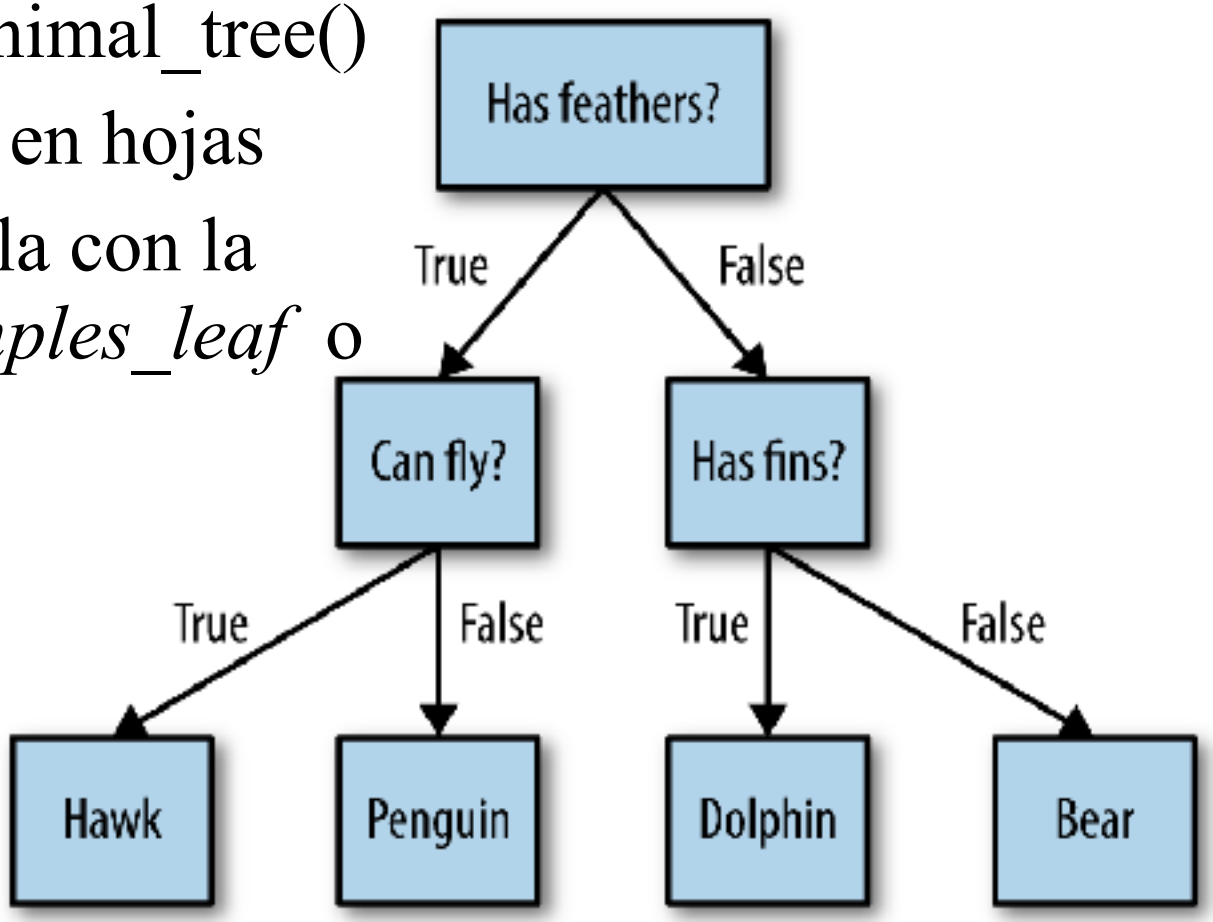
- Principal parámetro es la regularización
- Alto valor de α o bajo valor de C -> modelos más simples
- Si pocas características son importantes use regularización L1.
- Pocas características son más simples de explicar.
- Modelos lineales son rápidos para entrenar y predecir.
- Para bases datos muy grandes solver=´sag´
-

Naive Bayes

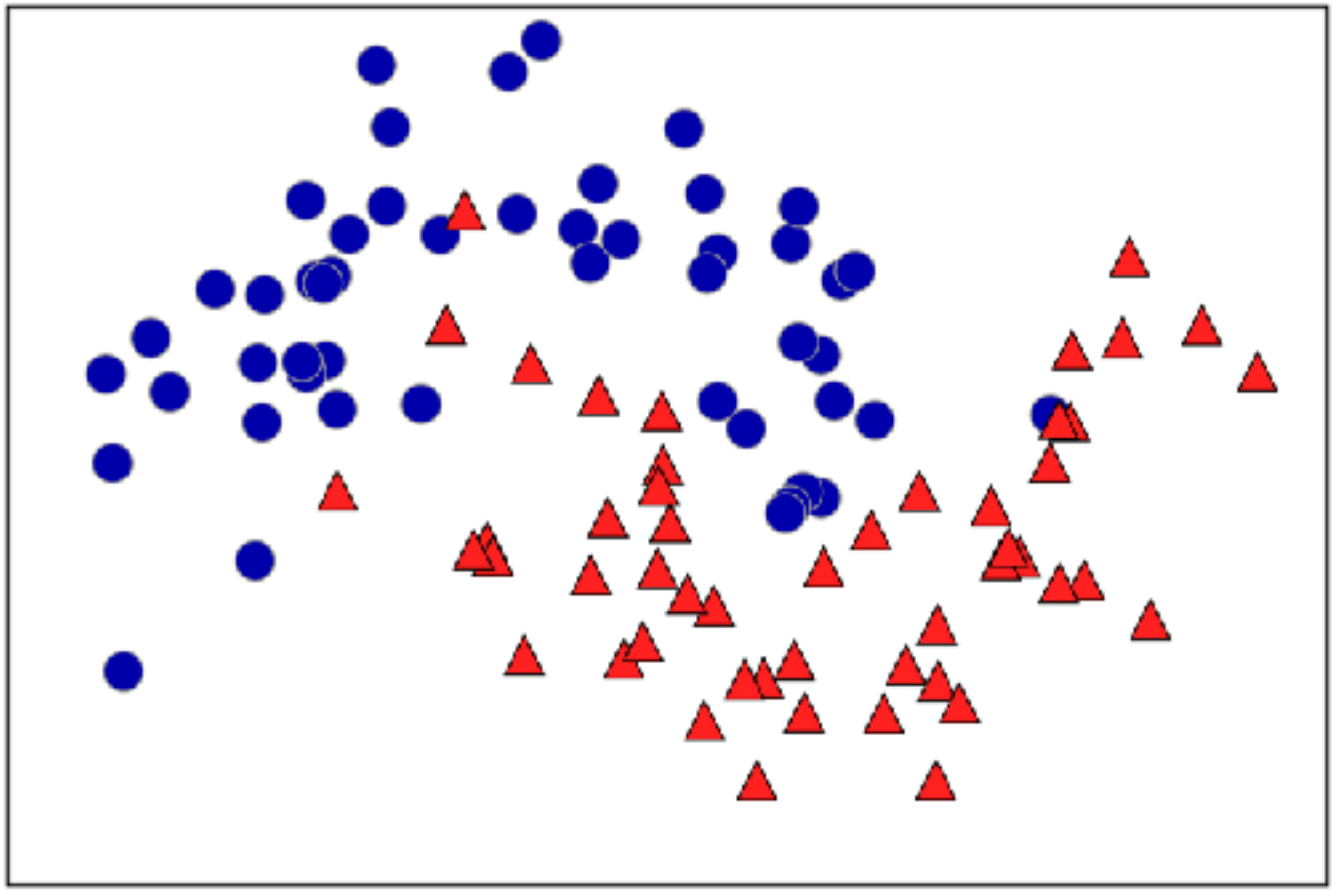
- Rápidos en entrenamiento
- Desempeño en generalización inferior a modelos lineales
- Utiles para datasets grandes de múltiples dimensiones
- Usan un parámetro α
- Tipos de Naive Bayes
  - GaussianNB - datos continuos
  - BernoulliNB - datos binarios
  - MultinomialNB - cuentas enteros

Arboles de decisión

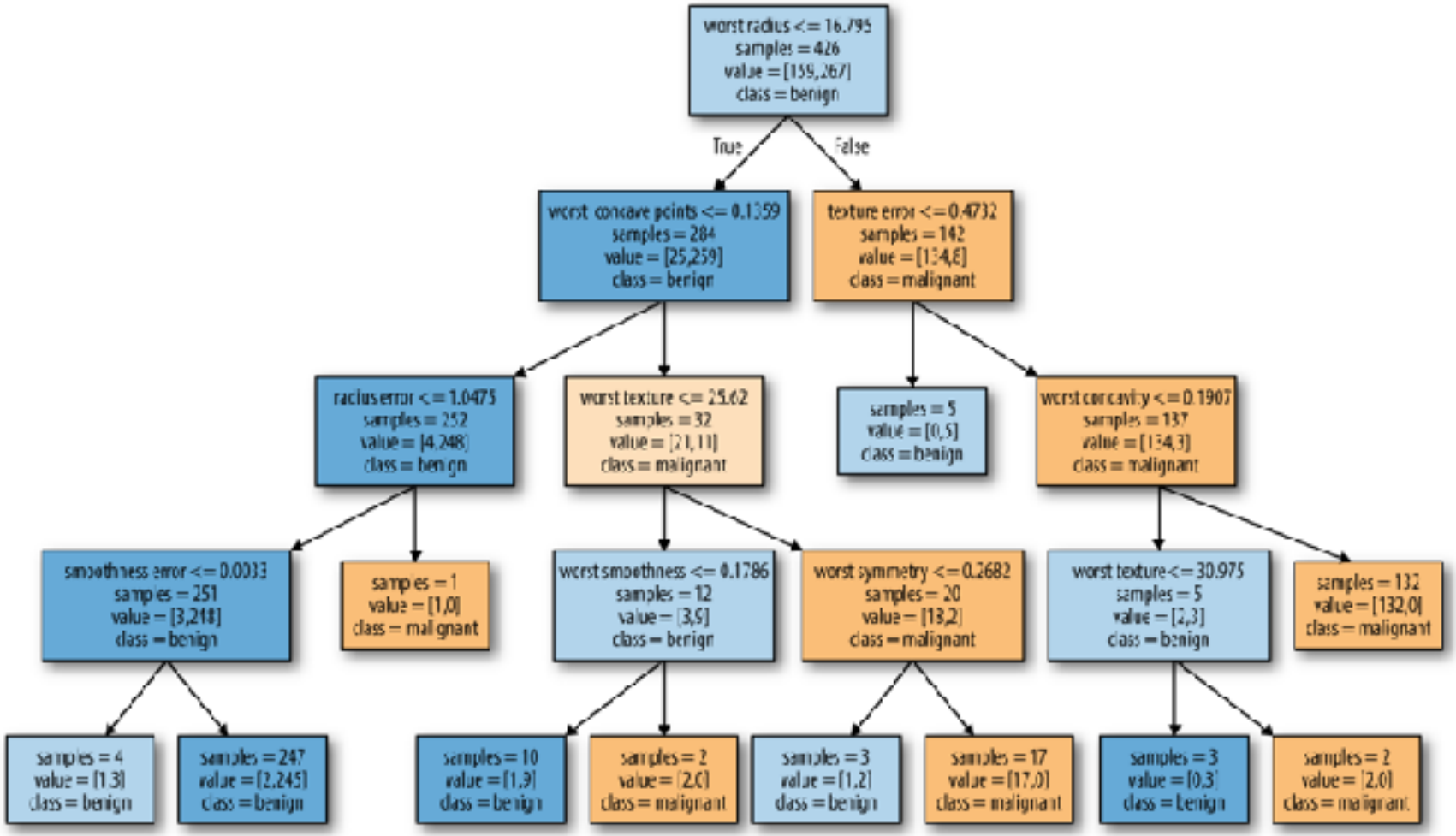
- Estructura if/else
- mglearn.plots.plot_animal_tree()
- Preguntas o terminal en hojas
- Overfitting se controla con la *max_depth*, *min_samples_leaf* o *max_leaf_nodes*
- Se pueden analizar
- Random forest



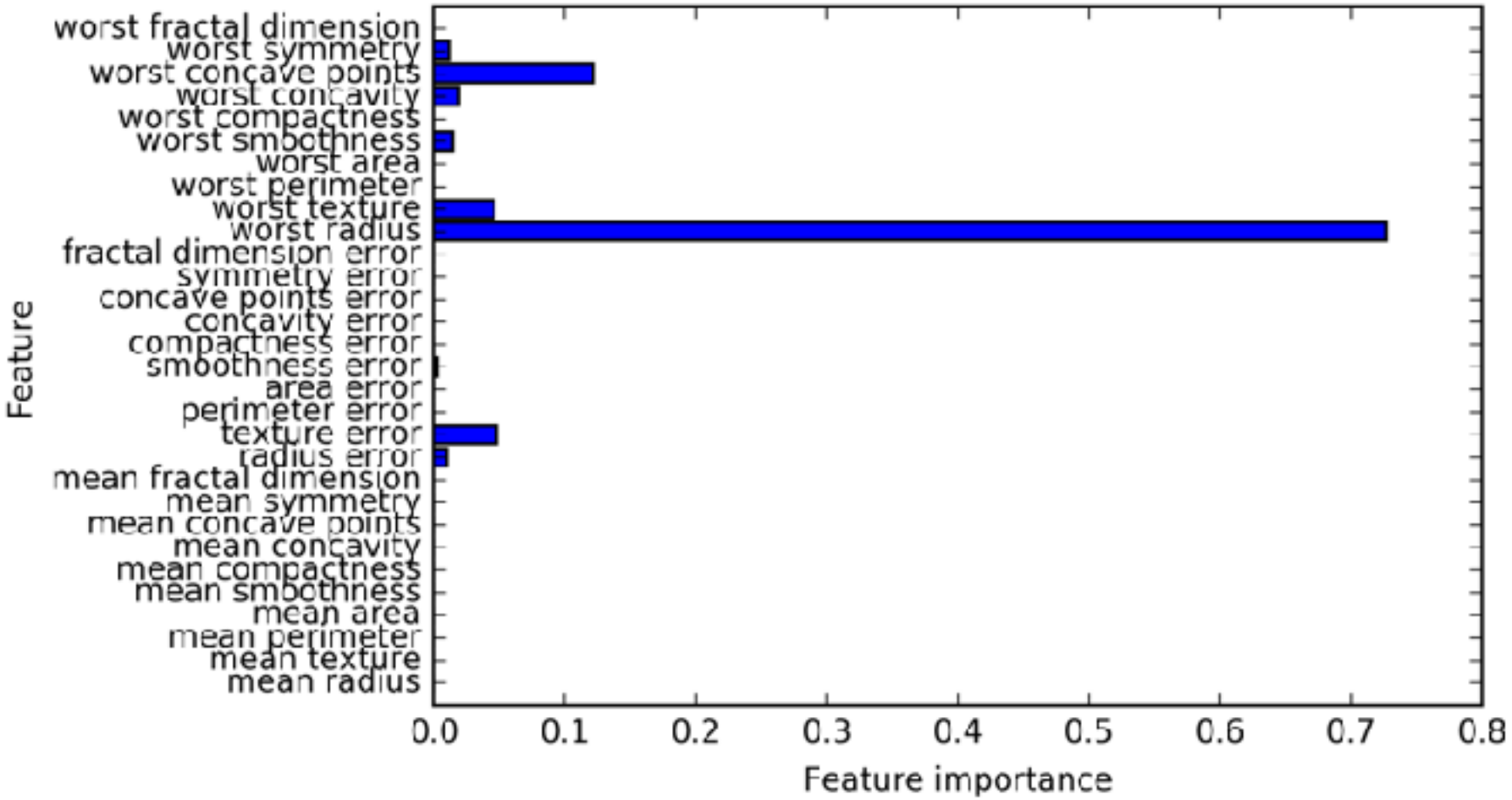
Arboles de decisión



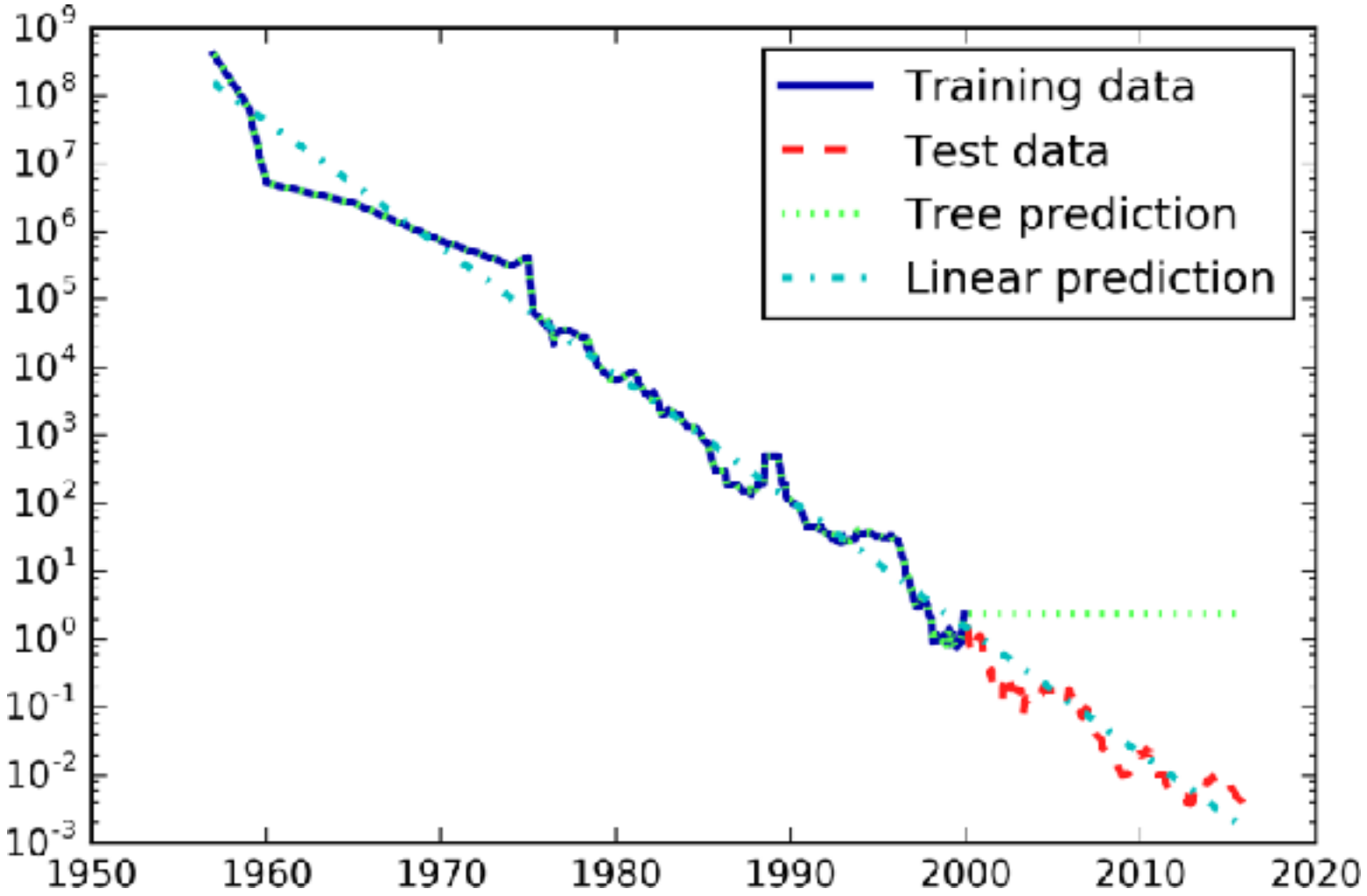
Análisis del árbol de decisiones



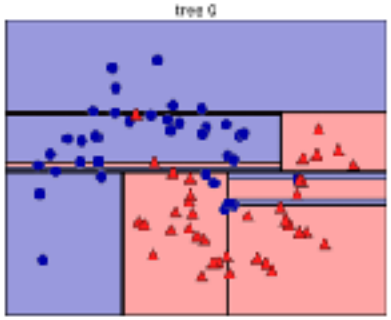
Importancia de las características

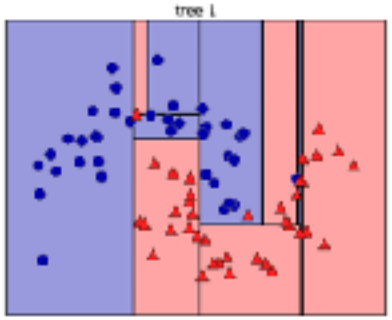


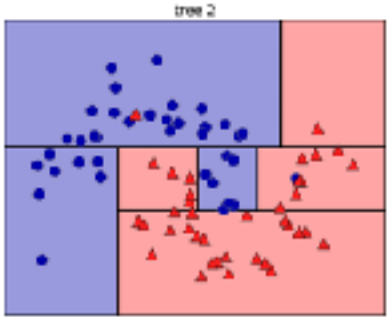
Comparación

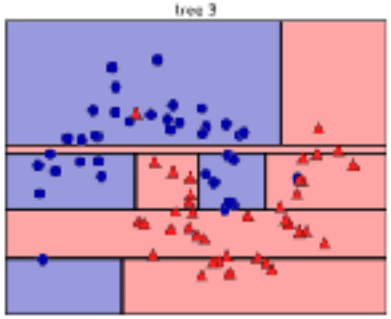


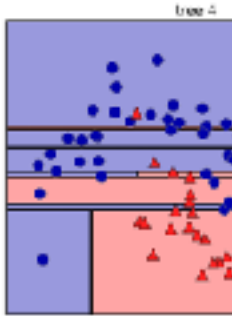
Random forest

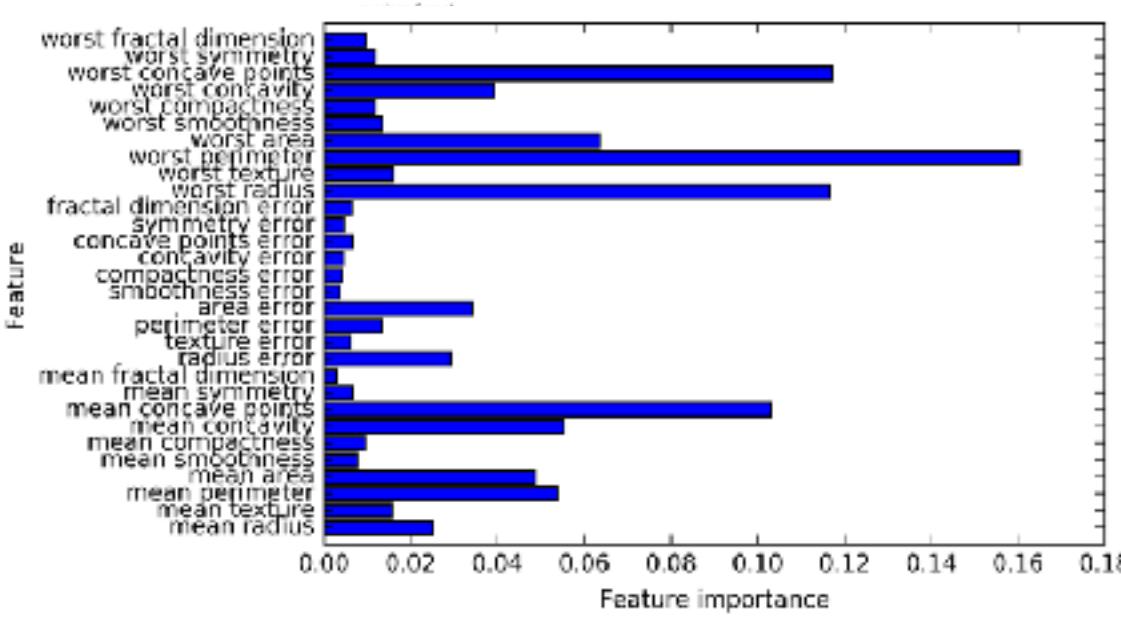








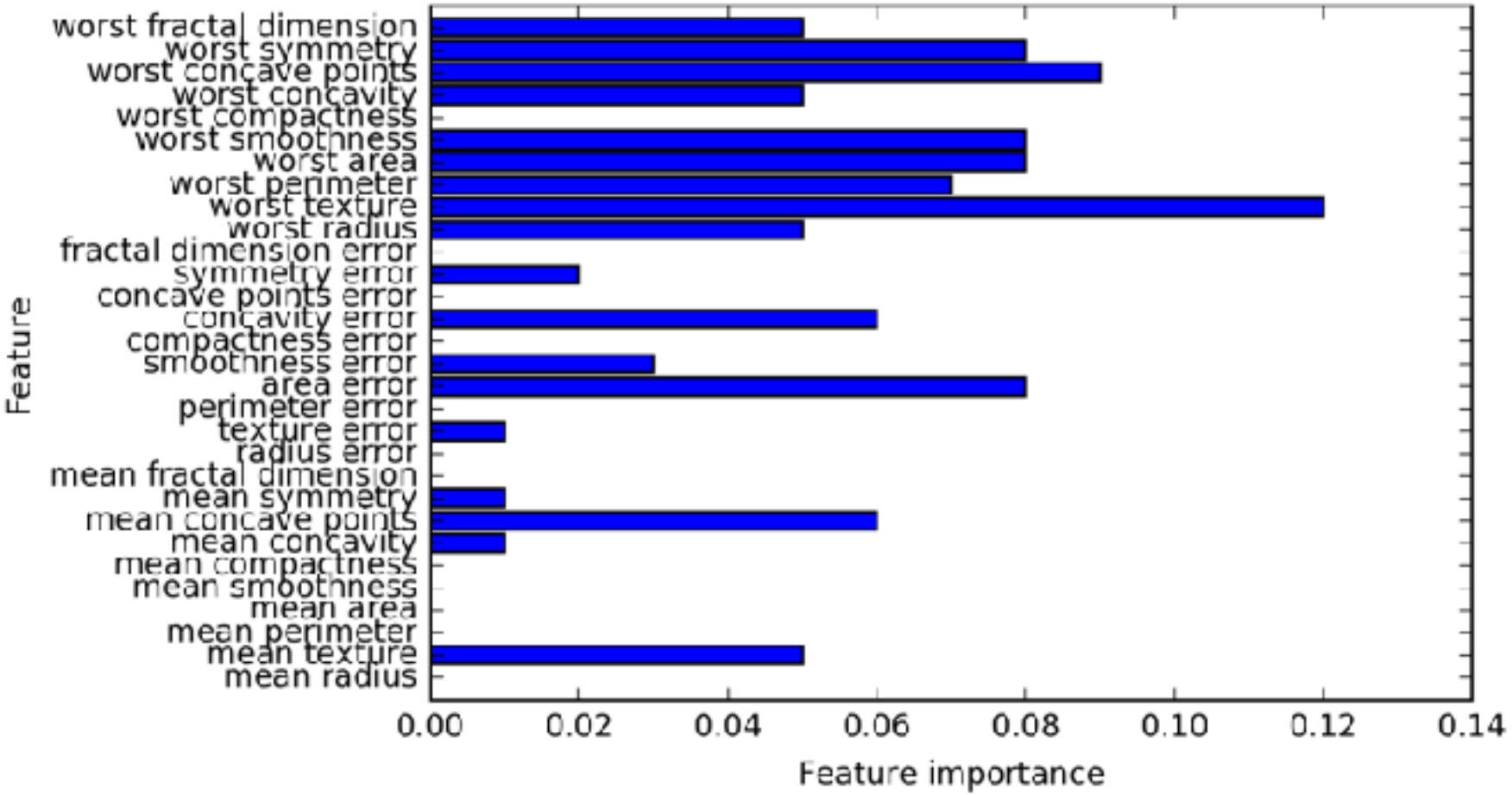




Gradient boosted regression

- Construye árboles serialmente
- Cada árbol trata de corregir errores de los anteriores
- Poca profundidad - 1 a 5 lo cual requiere menos memoria y mejora desempeño.
- Arboles agregados iterativamente mejoran desempeño.
- Se controla *learning_rate* que tan fuerte se corrigen los errores anteriores. Un valor alto hace el modelo más complejo
- Desempeño es superior al random forest
- Toma tiempo para entrenar
- Sensible a los parámetros
  - Número de árboles *n_estimators*
  - *learning_rate*

Features


worst fractal dimension
worst symmetry
worst concave points
worst concavity
worst compactness
worst smoothness
worst area
worst perimeter
worst texture
worst radius
fractal dimension error
symmetry error
concave points error
concavity error
compactness error
smoothness error
area error
perimeter error
texture error
radius error
mean fractal dimension
mean symmetry
mean concave points
mean concavity
mean compactness
mean smoothness
mean area
mean perimeter
mean texture
mean radius
Feature
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
Feature importance

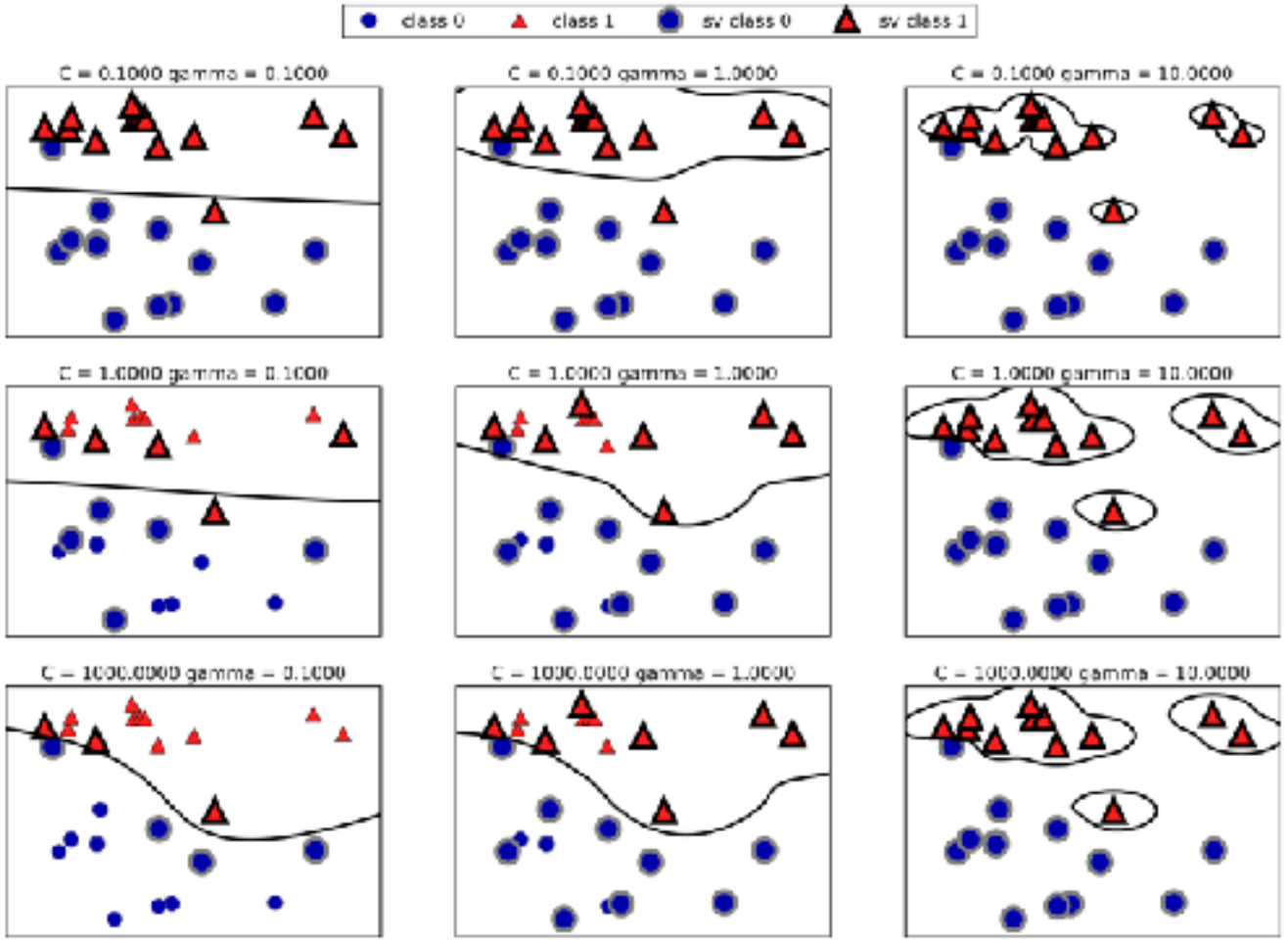
Support Vector Machines - SVM

- Los datos del borde de decisión son más importantes. Son el *support vector*
- Para hacer una predicción, la distancia a cada vector del punto es medida.
- $k_{\text{rbf}}(x_1, x_2) = \exp(\gamma \|x_1 - x_2\|^2)$
- La importancia de los vectores es el **aprendizaje**
- γ ancho de la gausiana

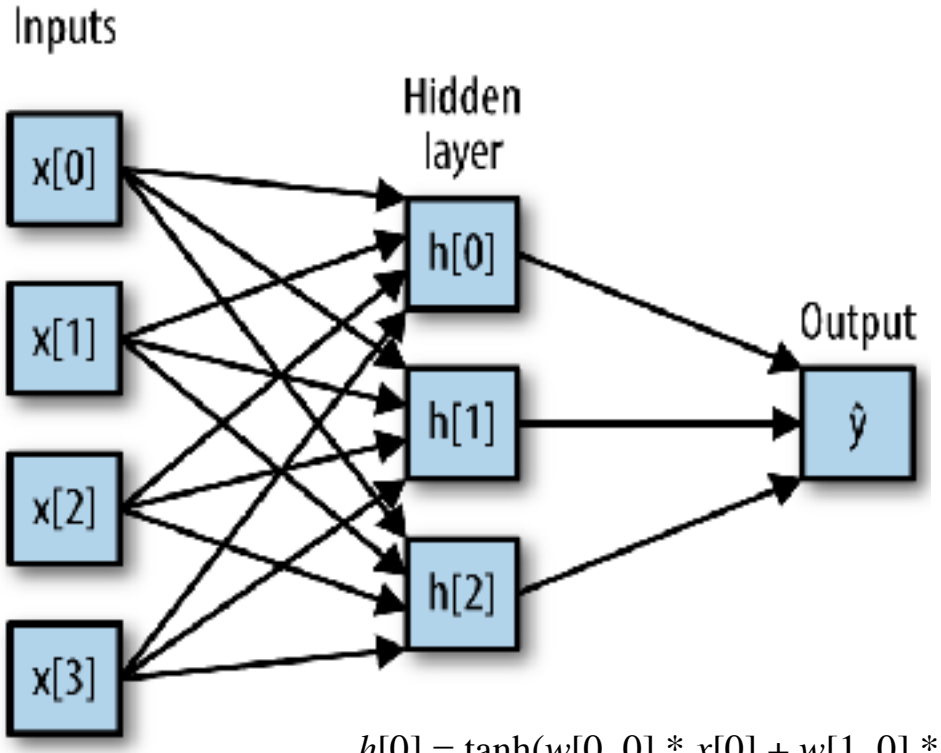
C y Gamma (γ) en SVM

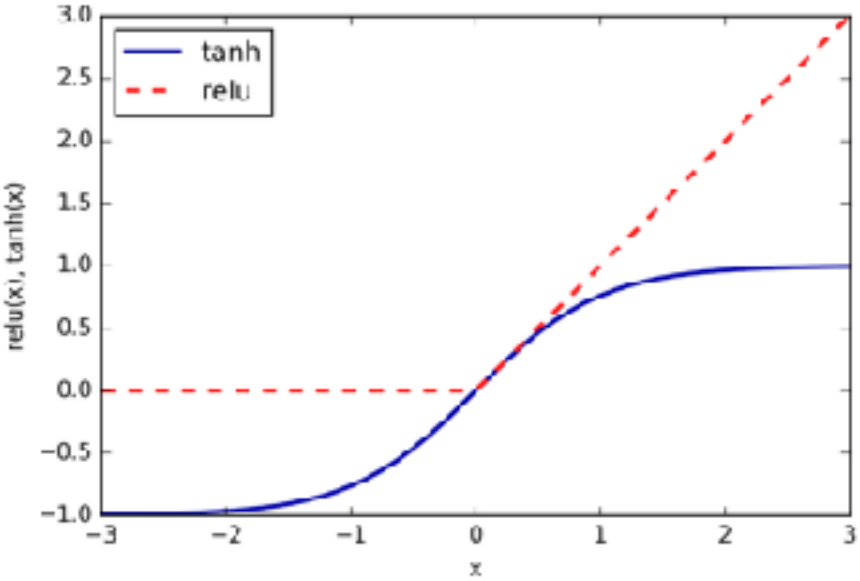
- *Gamma* limita el ancho del Gaussiano
- *Gamma* pequeño significa un radio grande y muchos puntos cercanos son considerados.
- *Gamma* bajo produce un modelo menos complejo
- C es el parámetro de regularización. Limita la importancia de cada punto.
- C más alto hace que el borde de decisión los clasifique correctamente.

Support Vector Machines (SVM)



Redes neuronales





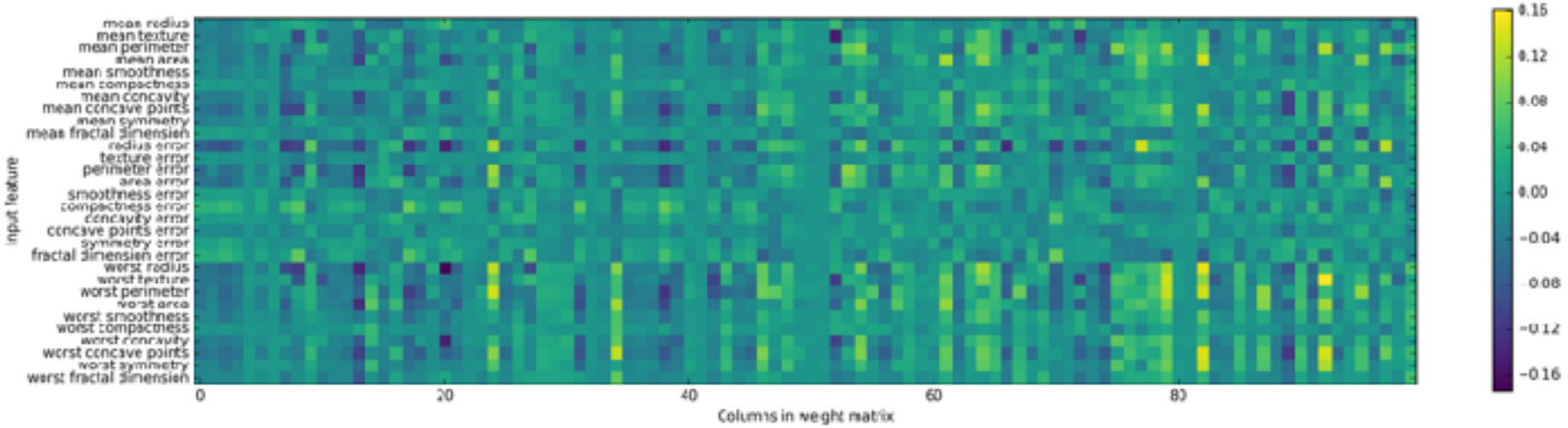
$$h[0] = \tanh(w[0,0] * x[0] + w[1,0] * x[1] + w[2,0] * x[2] + w[3,0] * x[3])$$

$$h[1] = \tanh(w[0,0] * x[0] + w[1,0] * x[1] + w[2,0] * x[2] + w[3,0] * x[3])$$

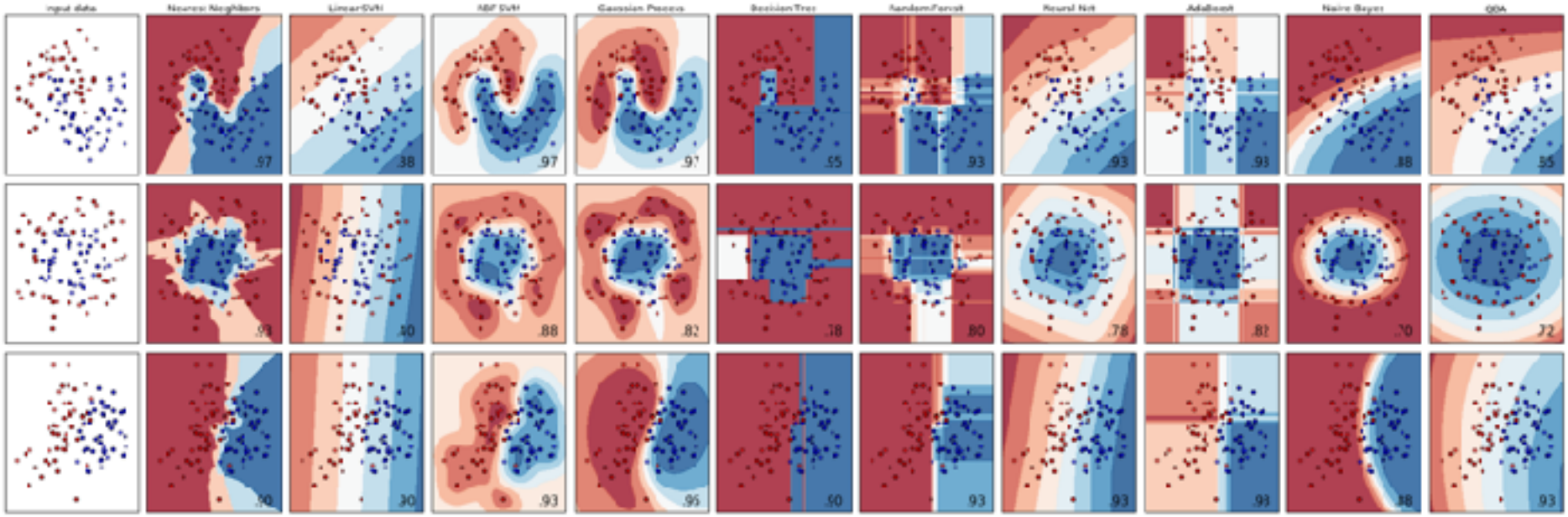
$$h[2] = \tanh(w[0,0] * x[0] + w[1,0] * x[1] + w[2,0] * x[2] + w[3,0] * x[3])$$

$$\hat{y} = v[0] * h[0] + v[1] * h[1] + v[2] * h[2]$$

Redes neuronales



Comparación de modelos



Comparación

- Nearest neighbors
- Linear models
- Naive Bayes
- Decision trees
- Random forests
- Gradient boosted decision trees
- Support vector machines
- Neural networks