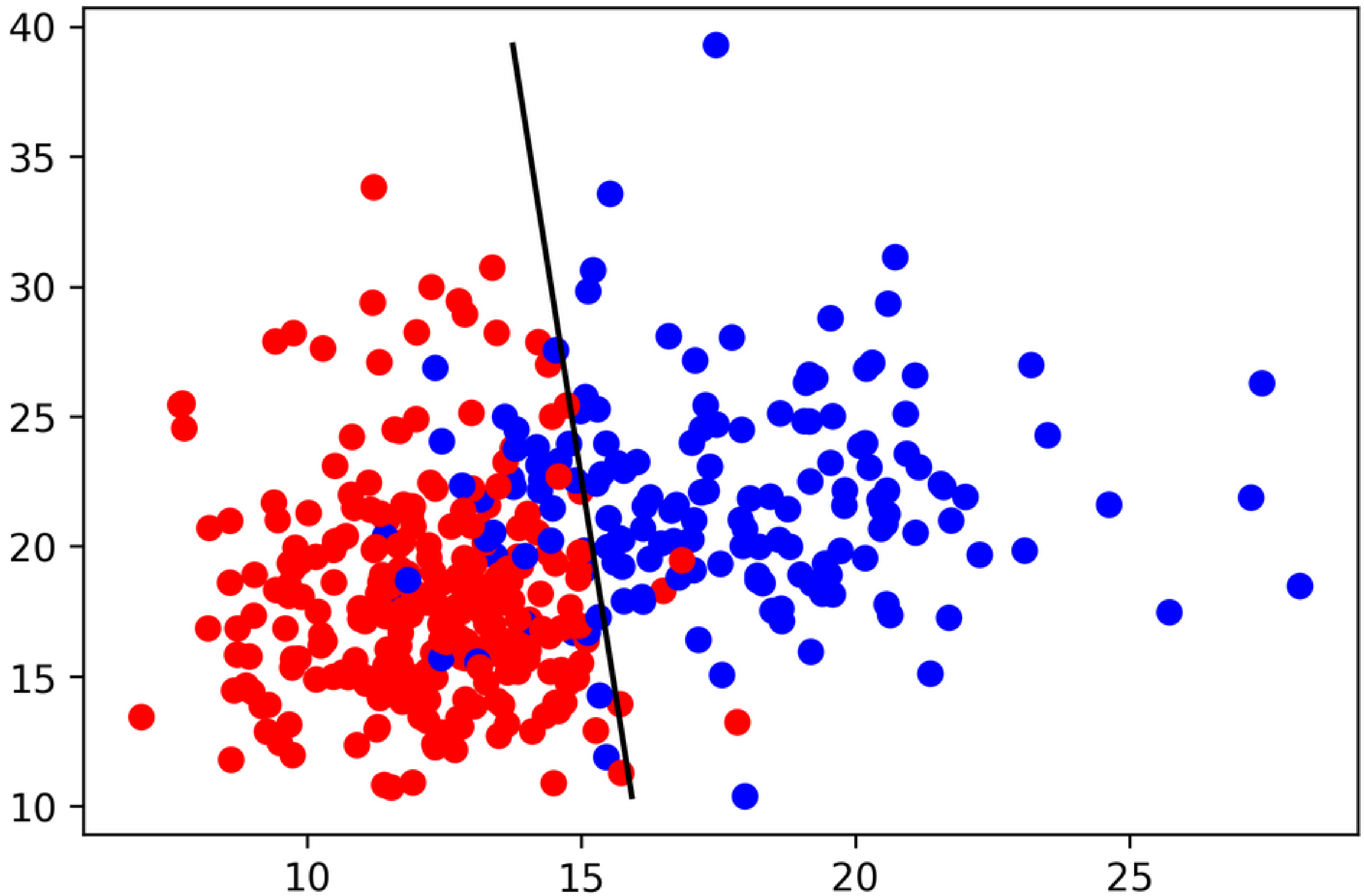
W4995 Applied Machine Learning

Linear Models for Classification

02/06/17

Andreas Müller

Linear models for **binary** classfiication

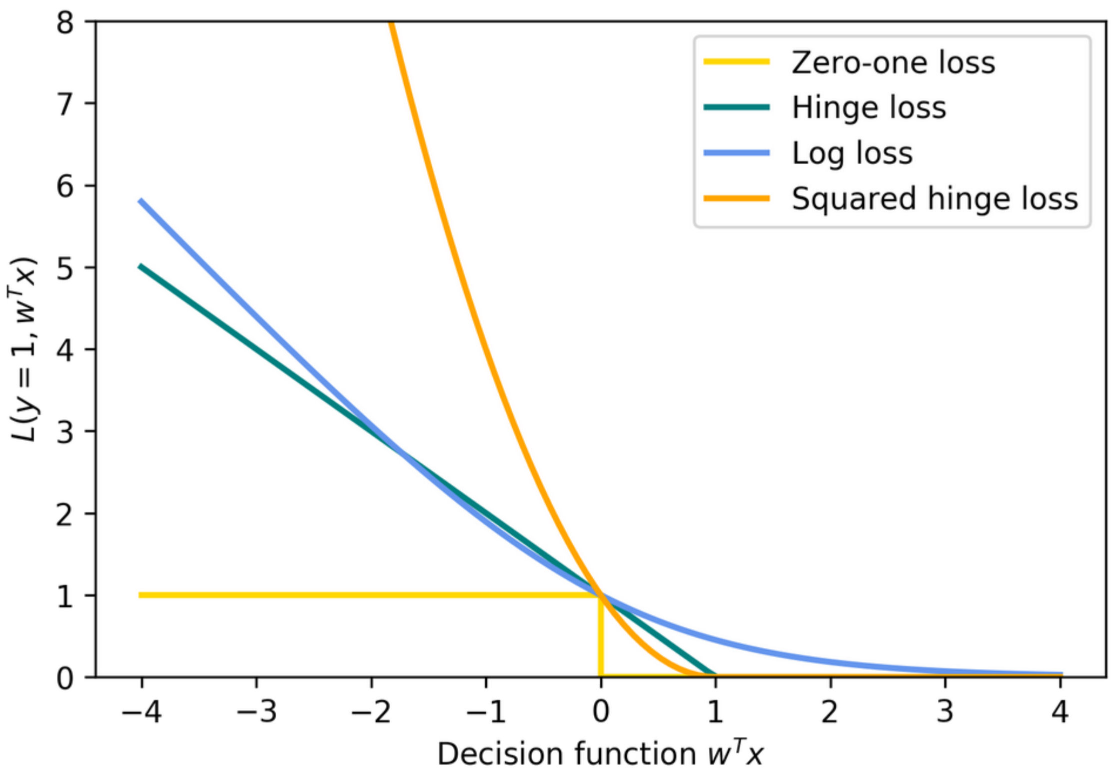


$$\hat{y} = \mathrm{sign}(w^T \mathbf{x} + b) = \mathrm{sign}(\sum_i w_i x_i + b)$$

Picking a loss?

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

Obvious idea:
Minimize number of misclassifications aka 0-1 loss.
But: non-convex, not continuous => Relax

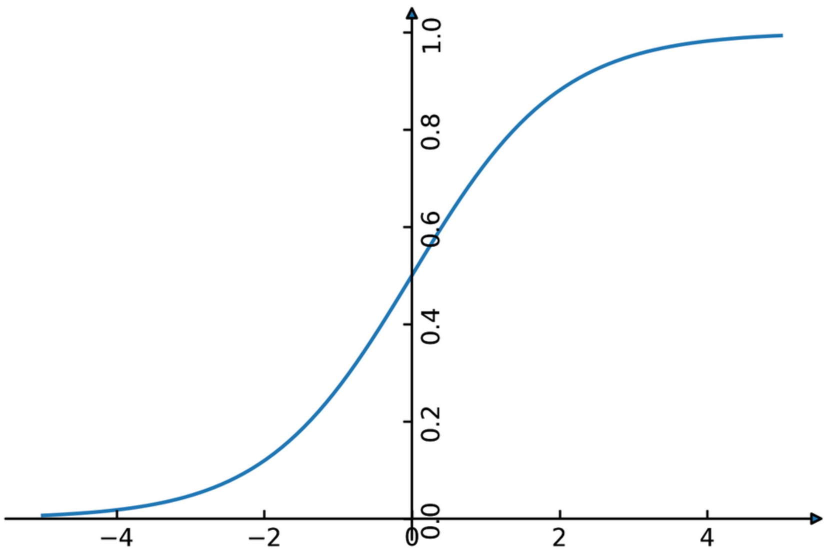


Logistic Regression

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

$$\log(\frac{p}{1-p}) = w^T \mathbf{x}$$

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



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

Penalized Logistic Regression

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

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

C is inverse to alpha (or alpha / n_samples)

Both versions strongly convex, l2 version smooth (differentiable).
All points contribute to w (dense solution to dual).

(soft margin) linear SVM

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

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

Both versions strongly convex, neither smooth.
Only some points contribute (the support vectors) to w (sparse solution to dual).

SVM or LogReg?

Do you need probability estimates?

no

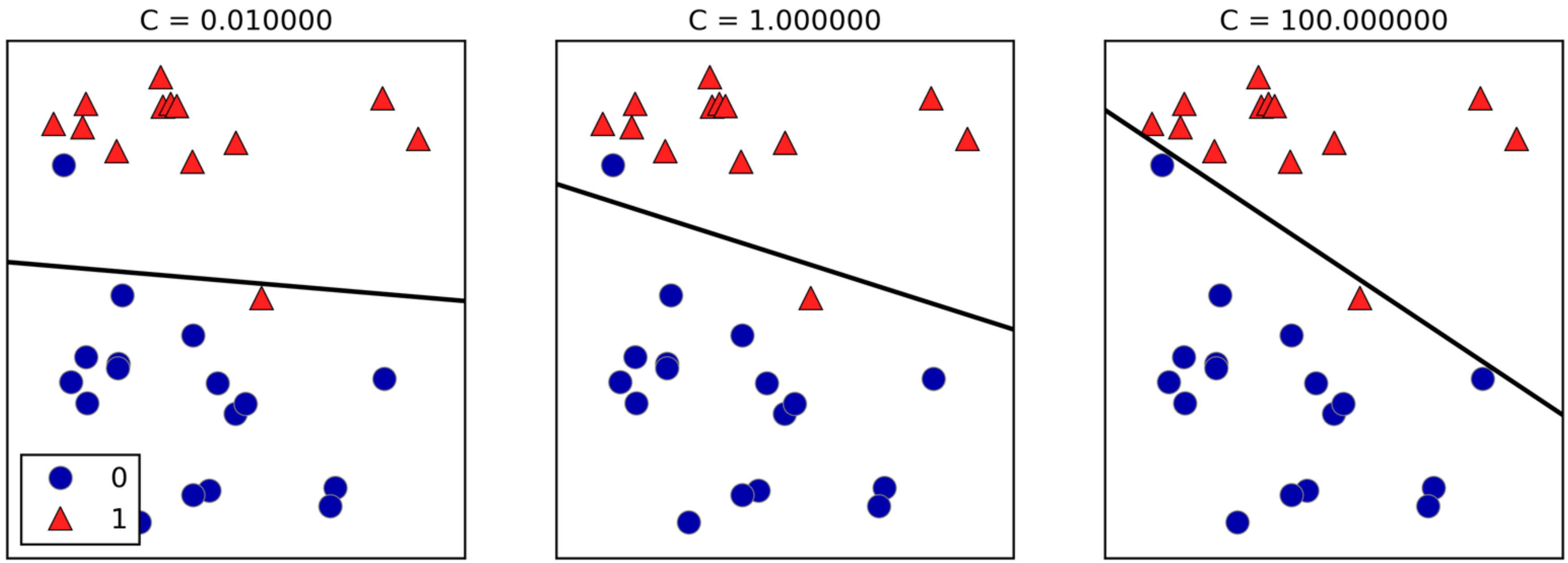
yes

It doesn't matter
- try either / both

Logistic Regression

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

Effect of regularization



Small C (little regularization) limits the influence of individual points!

Multiclass classification

Reduction to Binary classification

For 4 classes

- **One vs Rest**

Standard

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

n binary classifiers - each on all data

- **One vs One**

1v1, 1v2, 1v3, 1v4, 2v3, 2v4, 3v4

n * (n-1) / 2 binary classifiers - each on a fraction of the data

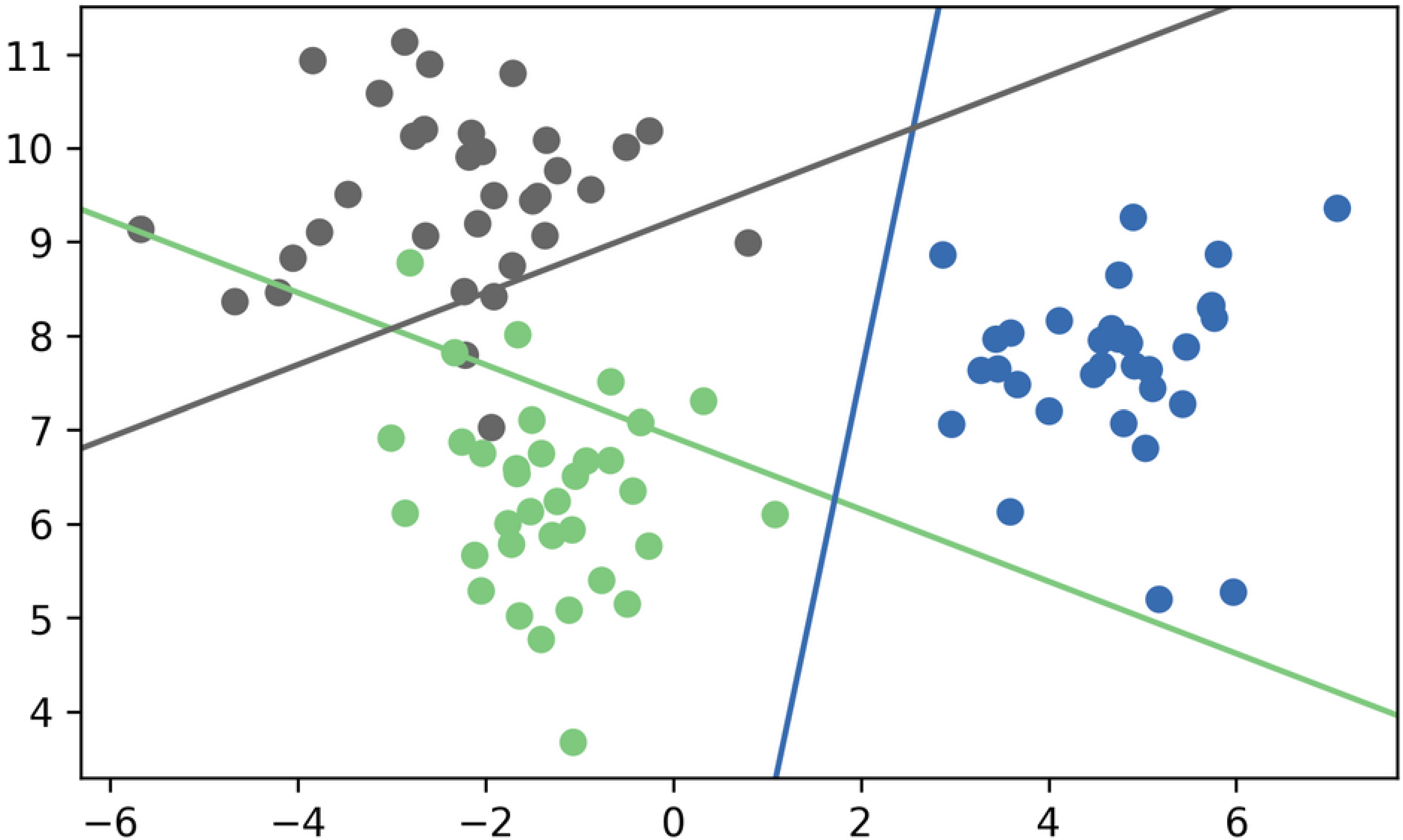
Prediction with One Vs Rest

- “Class with highest score”

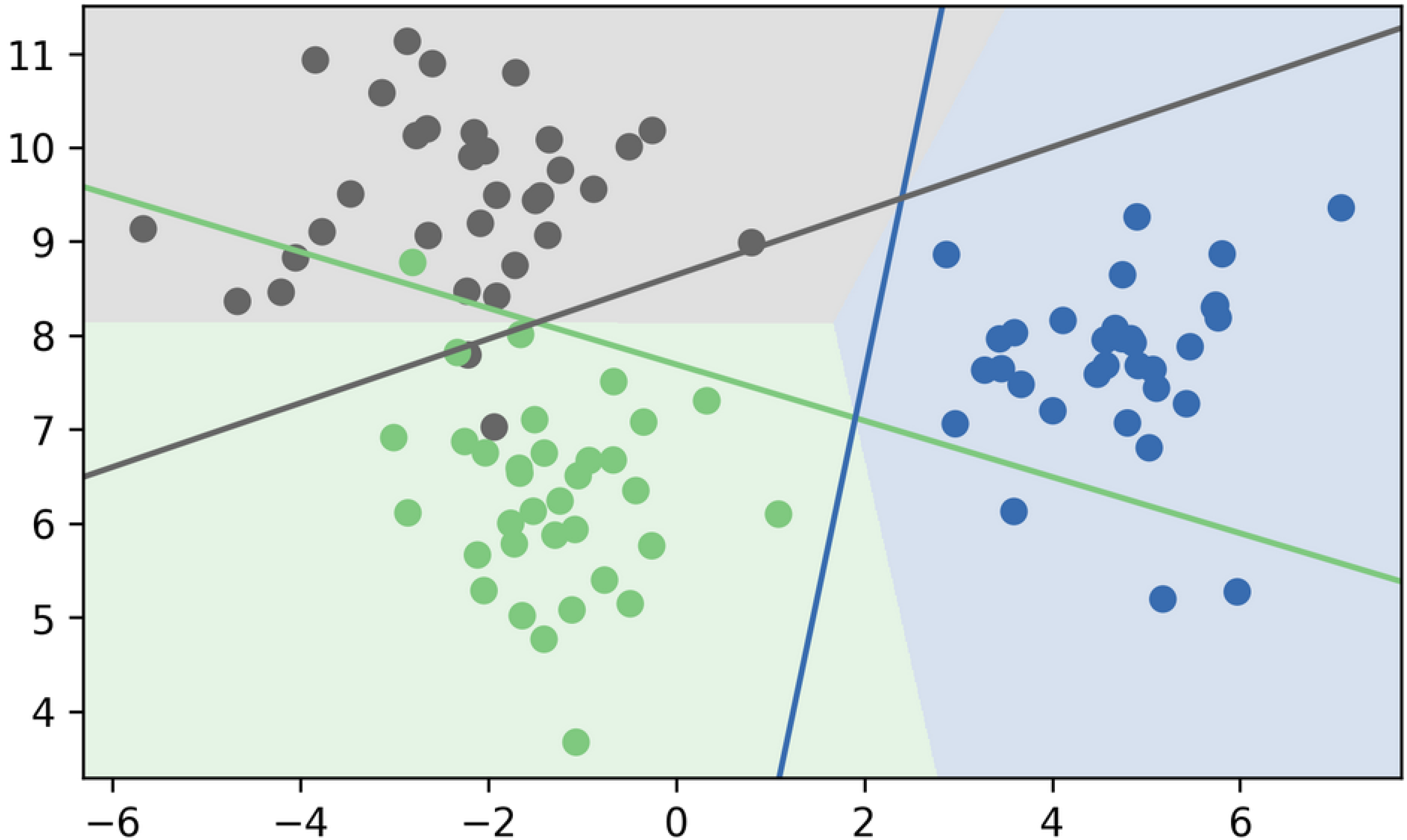
$$\hat{y} = \arg\max_{i \in Y} \mathbf{w}_i \mathbf{x}$$

- Unclear why it even works, but work well.

One vs Rest Prediction



One vs Rest Prediction

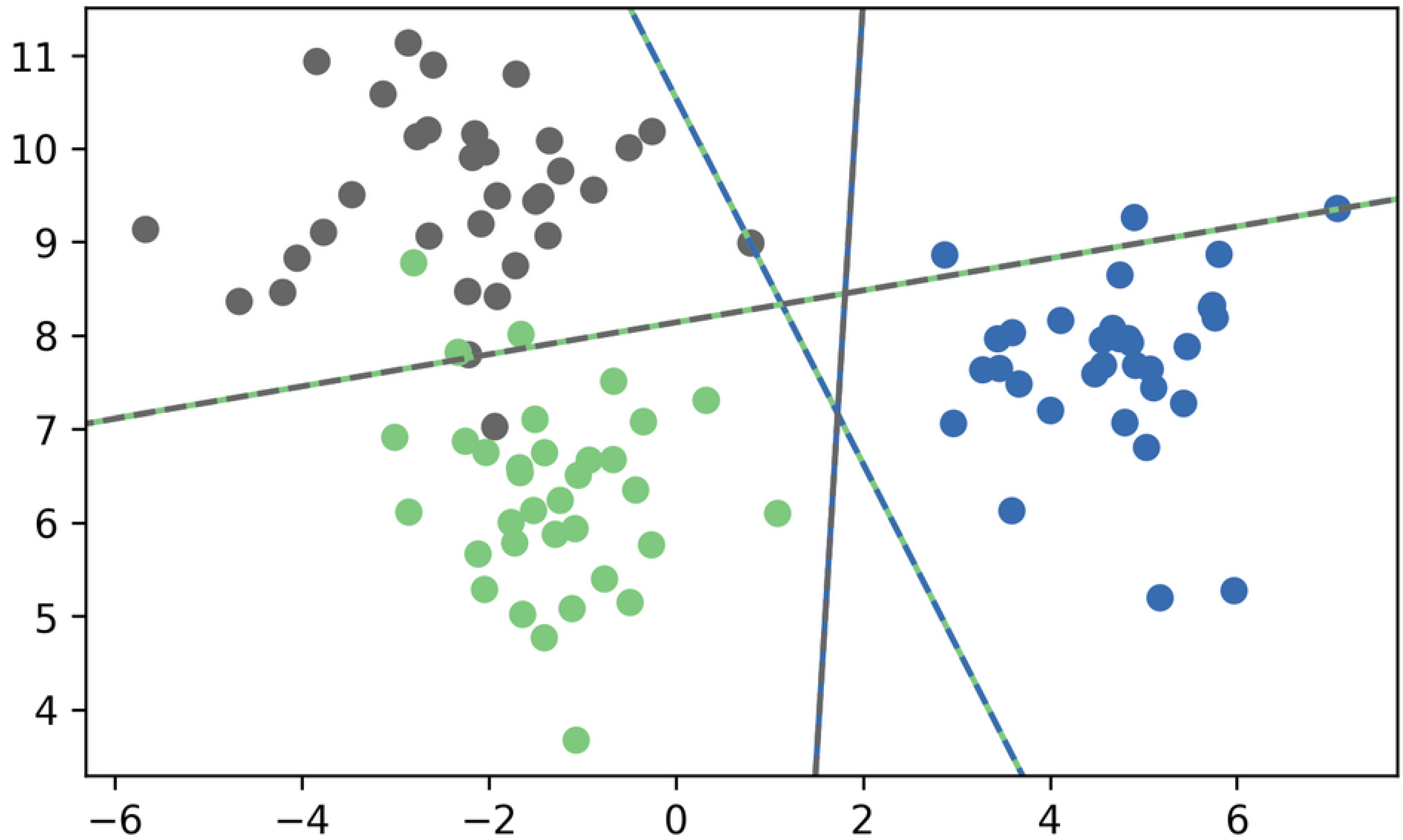


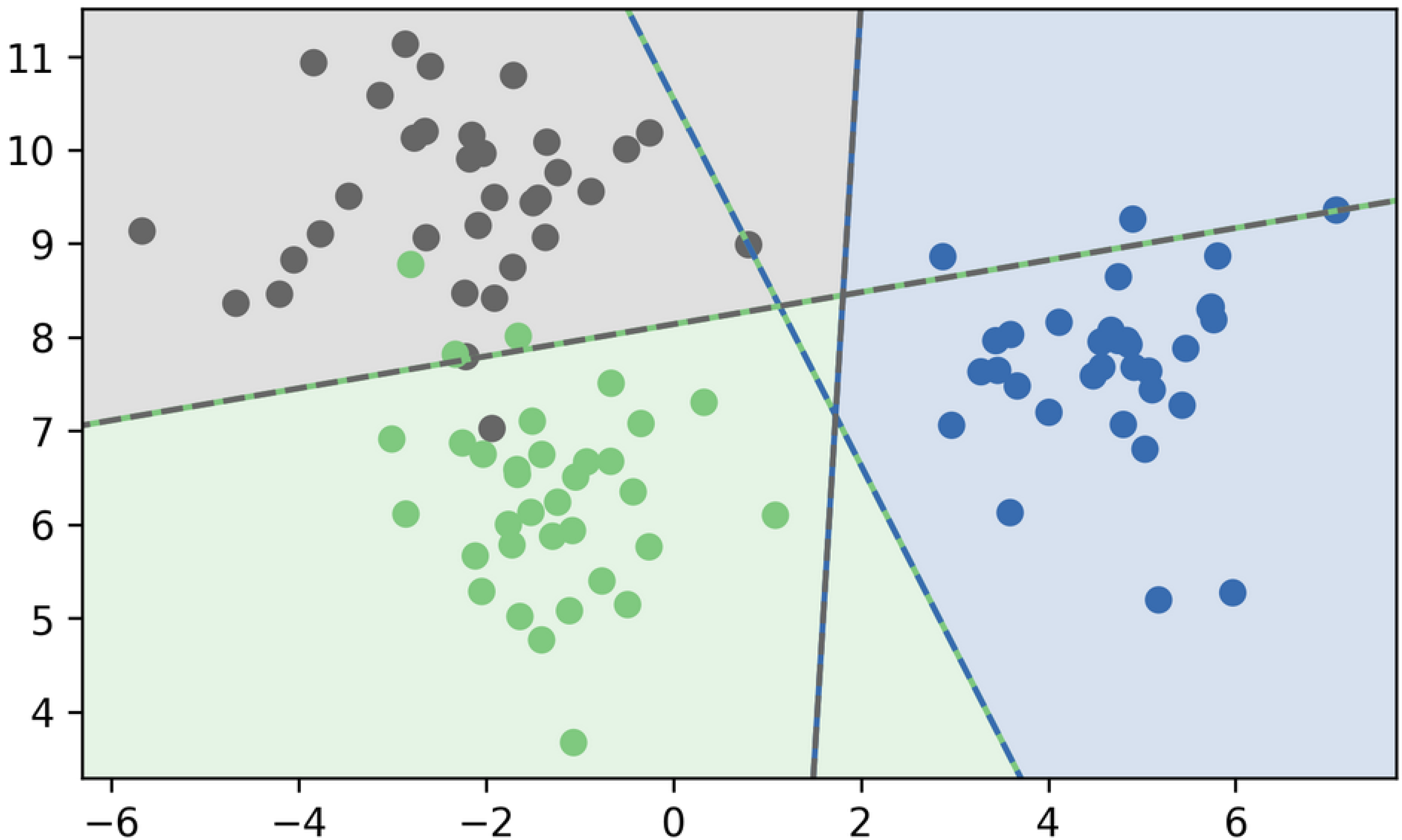
Prediction with One Vs One

- “Vote for highest positives”
- Classify by all classifiers.
- Count how often each class was predicted.
- Return most commonly predicted class.

- Again – just a heuristic.

One vs One Prediction




11
10
9
8
7
6
5
4
−6
−4
−2
0
2
4
6

Multinomial Logistic Regression

Probabilistic multi-class model:

$$p(y = i|\mathbf{x}) = \frac{e^{-\mathbf{w}_i^T \mathbf{x}}}{\sum_{j \in Y} e^{-\mathbf{w}_j^T \mathbf{x}}}$$

$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} \log(p(y = y_i|\mathbf{x}_i))$$

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

Same prediction rule as OvR!

In scikit-learn

- OvO: only SVC
- OvR: default for all linear models, even LogisticRegression
- LogisticRegression(multinomial=True)
- clf.decision_function = w^Tx
- logreg.predict_proba
- SVC(probability=True) not great

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]
```

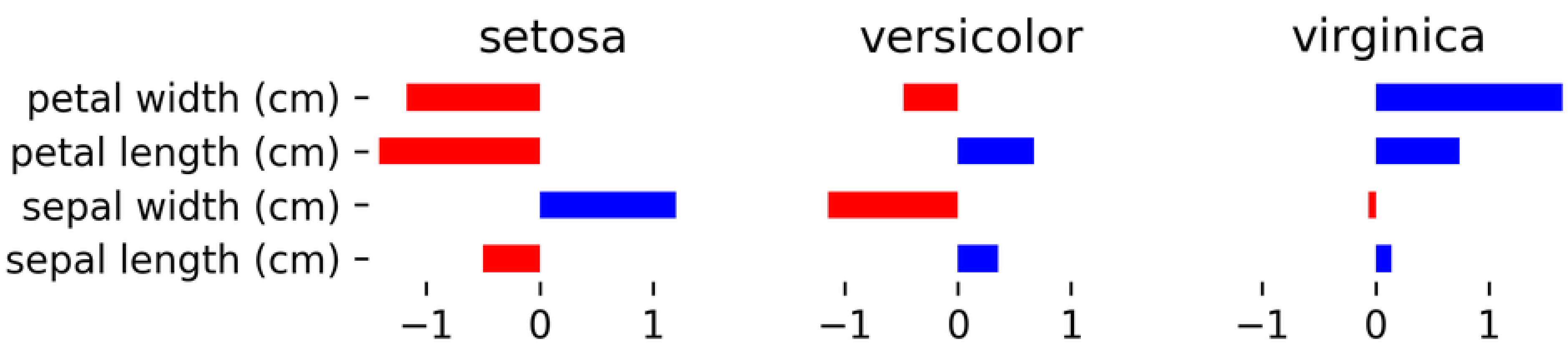
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC

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)

Solver choices

- Don't use SVC(kernel='linear'), use LinearSVC
- For n_features >> n_samples: Lars (or LassoLars) instead of Lasso.
- For small n_samples (<10.000?), don't worry.
- LinearSVC, LogisticRegression: dual=False if n_samples >> n_features
- LogisticRegression(solver="sag") for n_samples large.
- Stochastic Gradient Descent for "n_samples really large"

Efficient Cross-validation

Models with build-in cross-validation

- RidgeCV() - does GCV, approximation to LOO
- LarsCV(), LassoLarsCV(), ElasticNetCV()
- Use path-algorithms to compute the full solution path.
- LogisticRegressionCV() uses warm-starts, doesn't support all solvers.
- All have reasonable build-in parameter grids.
- For RidgeCV you can't pick the "cv"!

Using EstimatorCV

```
boston = load_boston()
X_train, X_test, y_train, y_test = train_test_split(
    boston.data, boston.target, random_state=42)

grid = GridSearchCV(Ridge(), param_grid={'alpha': np.linspace(.1, 1, 10)}, cv=10)
grid.fit(X_train, y_train)
print("Grid-search score: {:.2f}".format(grid.score(X_test, y_test)))
print("grid alpha: {}".format(grid.best_params_['alpha']))

ridge = RidgeCV().fit(X_train, y_train)
print("ridgecv score: {:.2f}".format(ridge.score(X_test, y_test)))
print("ridgecv alpha: {}".format(ridge.alpha_))
```

```
Grid-search score: 0.68
grid alpha: 0.1
ridgecv score: 0.68
ridgecv alpha: 0.1
```

Stochastic Gradient Descent

(see http://leon.bottou.org/projects/sgd
and http://leon.bottou.org/papers/bottou-bousquet-2008
and http://scikit-learn.org/stable/modules/scaling_strategies.html)

Decomposing Generalization Error

Error introduced by using
the training set instead of
"the true distribution"

$$\min_{\mathcal{F},\rho,n} \mathcal{E} = \mathcal{E}_{\text{app}} + \mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}} \quad \text{subject to} \begin{cases} n & \leq n_{\max} \\ T(\mathcal{F},\rho,n) & \leq T_{\max} \end{cases}$$

Error of restricting f
to family F

Error rho in finding
the f that's best on
the training set.

Bottou et. al.

Table 1: Typical variations when $\mathcal{F}$, n, and ρ increase.

		$\mathcal{F}$	n	ρ
$\mathcal{E}_{\mathrm{app}}$	(approximation error)	$\searrow$		
$\mathcal{E}_{\mathrm{est}}$	(estimation error)	$\nearrow$	$\searrow$	
$\mathcal{E}_{\mathrm{opt}}$	(optimization error)	$\cdots$	$\cdots$	$\nearrow$
T	(computation time)	$\nearrow$	$\nearrow$	$\searrow$

The Trade-off

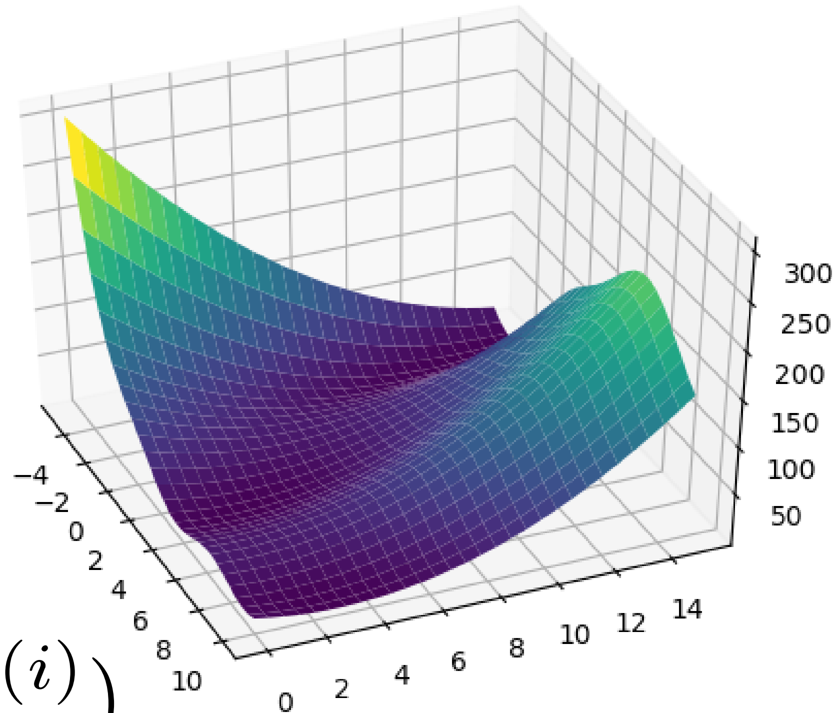
$$\min_{\mathcal{F},\rho,n} \quad \mathcal{E} = \mathcal{E}_{\text{app}} + \mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}} \quad \text{subject to} \begin{cases} n & \leq & n_{\max} \\ T(\mathcal{F},\rho,n) & \leq & T_{\max} \end{cases}$$

If n_max large, we are constraint by T_max!
Making the optimization error small doesn't matter if it means we can't look at all training examples!

Trade optimization error for estimation error
use a worse algorithm but look at more data!

Reminder: Gradient Descent

Want: $\arg\min_w F(w)$

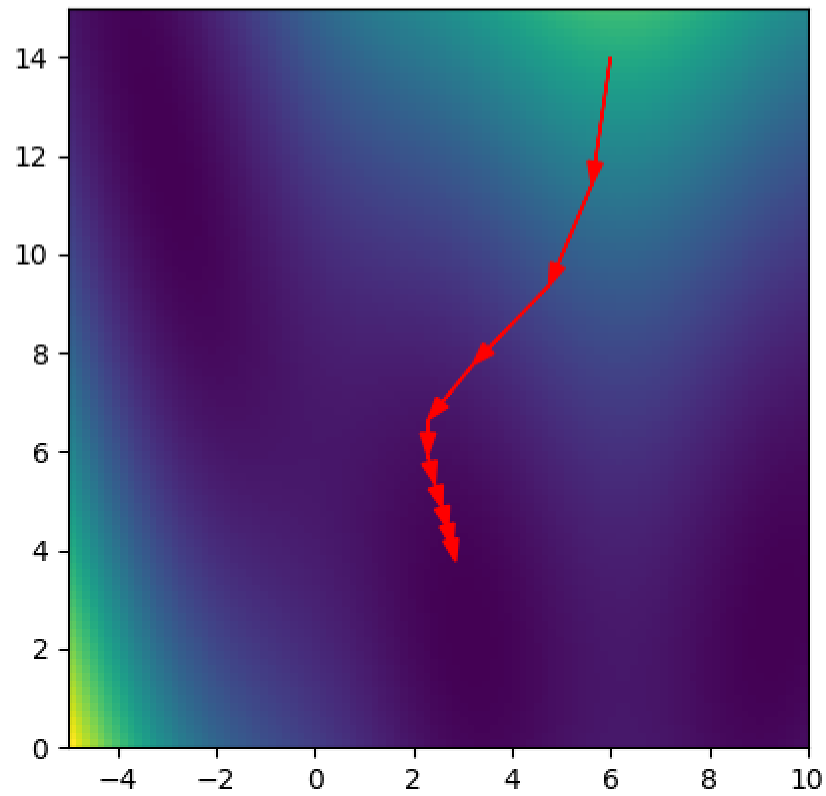


Initialize w_0

$$w^{(i+1)} \leftarrow w^{(i)} - \eta_i \frac{d}{dw} F(w^{(i)})$$

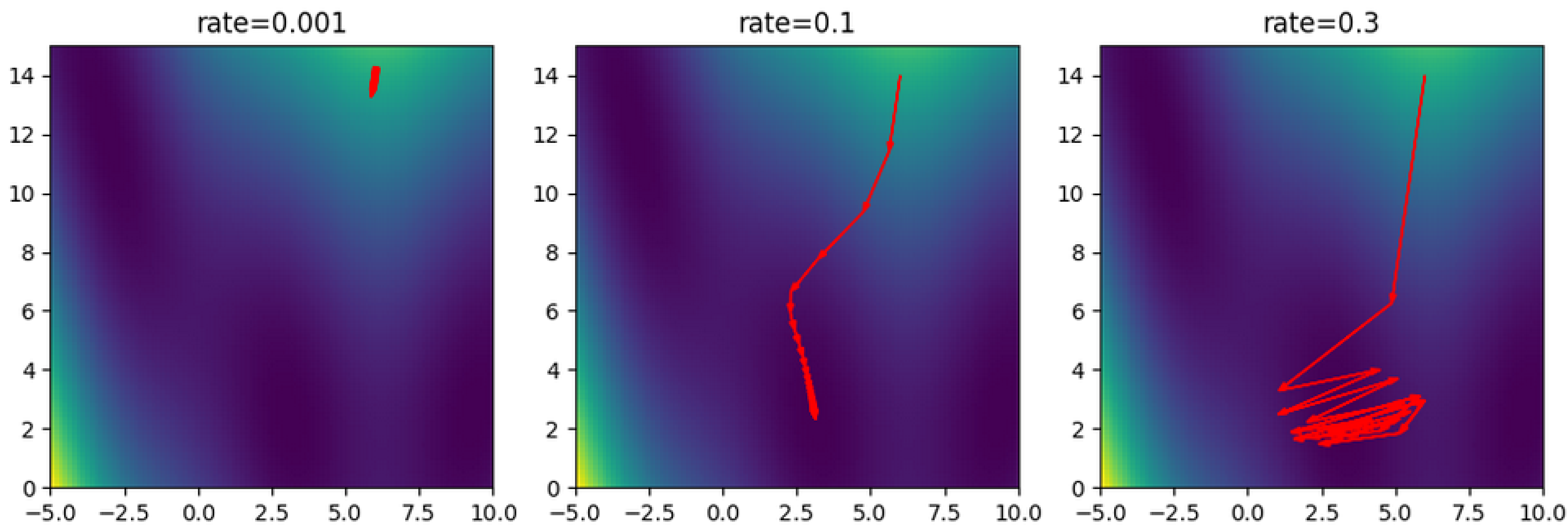
Converges to local minimum

Reminder: Gradient Descent



$$w^{(i+1)} \leftarrow w^{(i)} - \eta_i \frac{d}{dw} F(w^{(i)})$$

Picking a learning rate



$$w^{(i+1)} \leftarrow w^{(i)} - \eta_i \frac{d}{dw} F(w^{(i)})$$

Stochastic Gradient Descent

Logistic Regression:

$$F(w) = C \sum_{i=1}^{n} \log(\exp(-y_i w^T \mathbf{x}_i) + 1) + ||w||_2^2$$

Sum over data-points

Independent of data

Pick x_i randomly, then

$$\frac{d}{dw} F_i(w) = C \log(exp(-y_i w^T \mathbf{x}_i) + 1) + \frac{1}{n} ||w||_2^2$$

Is a stochastic approximation of gradient of F with expectation the actual gradient.

In practice: just iterate over i.

SGD and partial_fit

- SGDClassifier(), SGDRegressor() fast on very large datasets – have many different loss and regularization options.
- Tuning learning rate and schedule can be tricky. "optimal" learning rate only works with unit-variance data. Averaging can be helpful.
- partial_fit allows working with out-of-memory data!

```python
sgd = SGDClassifier()
for X_batch, y_batch in batches:
    sgd.partial_fit(X_batch, y_batch, classes=[0, 1, 2])
sgd.score(X_test, y_test)
```

```
0.8157894736842105l
```

```python
sgd = SGDClassifier()
for i in range(10):
    for X_batch, y_batch in batches:
        sgd.partial_fit(X_batch, y_batch, classes=[0, 1, 2])
sgd.score(X_test, y_test)
```

```
0.9473684210526315 3
```