3011979 Practical Python for Data Sciences and Machine Learning

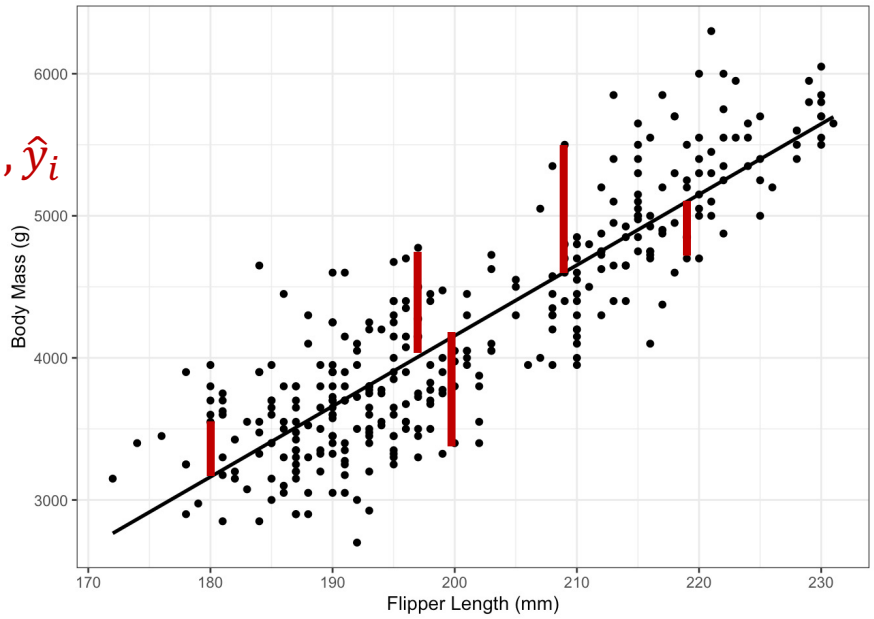
L7: Linear models

Feb 25th, 2022

Sira Sriswasdi, Ph.D.
Research Affairs, Faculty of Medicine
Chulalongkorn University

Linear regression

1D linear regression



Model

$$\hat{y} = f(x) = b_0 + b_1 x$$

Loss/Objective/Cost function

Mean Squared Error (MSE)

$$\sum_i (y_i - \hat{y}_i)^2 = \sum_i (y_i - [b_0 + b_1 x_i])^2$$

Optimal solution of linear regression that optimize MSE is simple!

- Minimize MSE with calculus
  - $\frac{\delta MSE}{\delta b_o} = \sum_i -2(y_i - [b_0 + b_1 x_i]) = -2(\sum_i y_i - b_1 \sum_i x_i - n b_0)$
    - Optimal at $b_0 = \bar{y} - b_1 \bar{x}$
  - $\frac{\delta MSE}{\delta b_1} = \sum_i -2x_i(y_i - [b_0 + b_1 x_i]) = -2\left(\sum_i x_i y_i - b_1 \sum_i x_i^2 - b_0 \sum_i x_i\right)$
    - Optimal at $b_0 = \frac{\sum_i x_i y_i - b_1 \sum_i x_i^2}{\sum_i x_i}$

Extension to multivariate case

Model

$$\hat{y} = f(x) = b_0 + b_1 x_1^{(i)} + b_2 x_2^{(i)} + \cdots + b_n x_n^{(i)}$$

MSE loss

$$\sum_i (y_i - \hat{y}_i)^2 = \sum_i \left(y_i - [b_0 + b_1 x_1^{(i)} + b_2 x_2^{(i)} + \cdots + b_n x_n^{(i)}] \right)^2$$

- Minimize MSE with calculus
  - $\frac{\delta MSE}{\delta b_o} = \sum_i -2\left(y_i - f(x)\right)$
  - $\frac{\delta MSE}{\delta b_1} = \sum_i -2x_1^{(i)}\left(y_i - f(x)\right)$
  - $\frac{\delta MSE}{\delta b_2} = \sum_i -2x_2^{(i)}\left(y_i - f(x)\right)$
  - …
  - $\frac{\delta MSE}{\delta b_n} = \sum_i -2x_n^{(i)}\left(y_i - f(x)\right)$

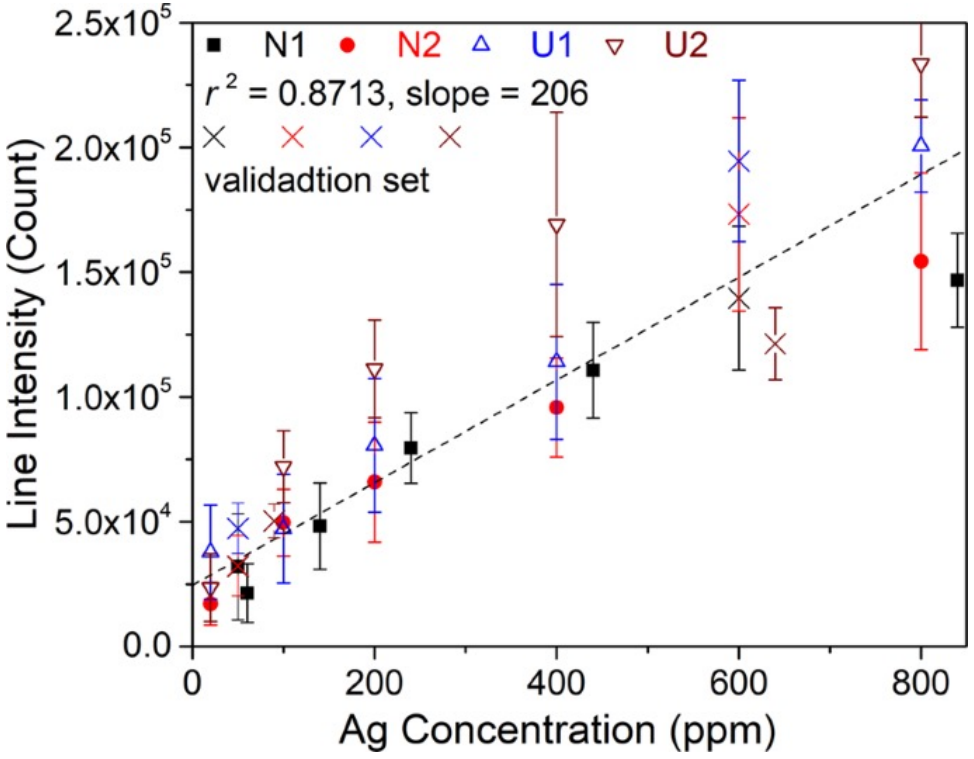
Why MSE? Ordinary Least Square (OLS)

- Measurements have errors
  - $y_i = b_0 + b_1 x_1^{(i)} + b_2 x_2^{(i)} + \cdots + b_n x_n^{(i)} + \varepsilon_i$ where ε_i is the error term

- Typical assumptions
  - ε_i doesn't correlate with other ε_j
  - ε_i has the same variance in every observation
  - ε_i is centered at zero

- Interpretation of MSE
  - MSE $= \sum_i (y_i - \hat{y}_i)^2 = \sum_i {\varepsilon_i}^2$
  - Minimize total variance of observation errors

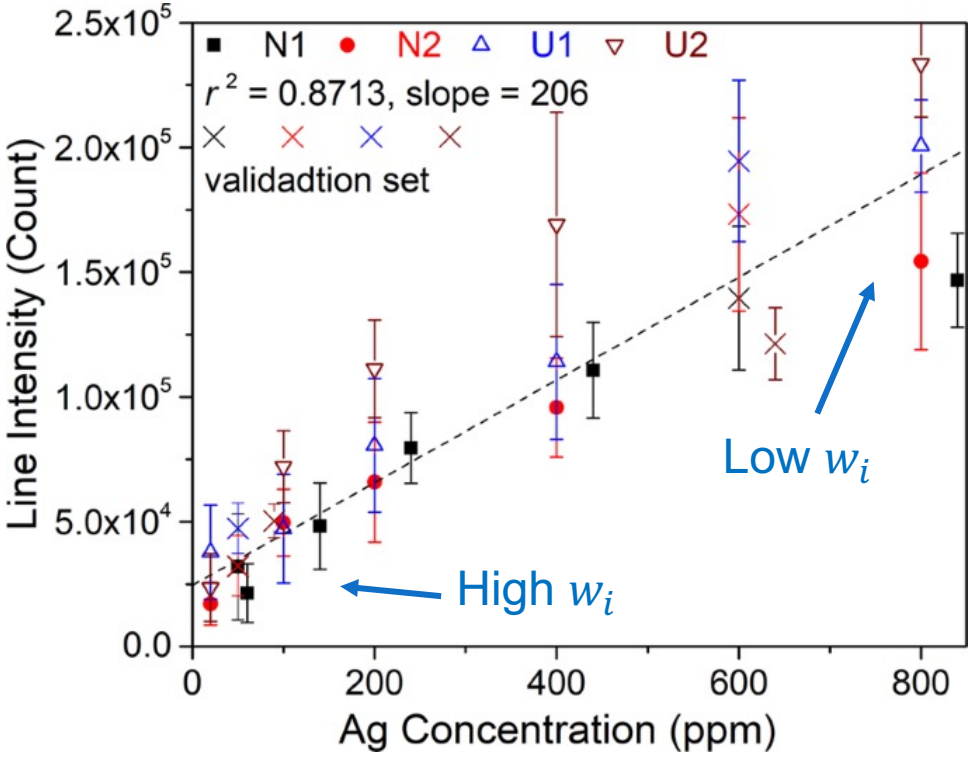
Extension of OLS



Sun, C. et al. "Machine Learning Allows Calibration Models to Predict Trace Element Concentration in Soils with Generalized LIBS Spectra" Scientific Reports (2019)

- Errors sometimes scale with the magnitude of input data x
- Prioritize fitting of certain regions of the input

Weighted least square



Sun, C. et al. "Machine Learning Allows Calibration Models to Predict Trace Element Concentration in Soils with Generalized LIBS Spectra" Scientific Reports (2019)

- Add weight w_i to each observation
- $\sum_i w_i(y_i - \hat{y}_i)^2 = \sum_i w_i \left(y_i - [b_0 + b_1 x_1^{(i)} + b_2 x_2^{(i)} + \cdots + b_n x_n^{(i)}]\right)^2$

Linear regression in Python

sklearn.linear_model.LinearRegression

class `sklearn.linear_model.` **`LinearRegression`**(*, *fit_intercept=True*, *normalize=False*, *copy_X=True*, *n_jobs=None*, *positive=False*) ¶ [source]

No random_state parameter

Ordinary least squares Linear Regression.

LinearRegression fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Parameters:

fit_intercept : *bool, default=True*

Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

normalize : *bool, default=False*

This parameter is ignored when `fit_intercept` is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, please use `StandardScaler` before calling `fit` on an estimator with `normalize=False`.

Logistic regression

Limitation of linear regression on {0,1} output

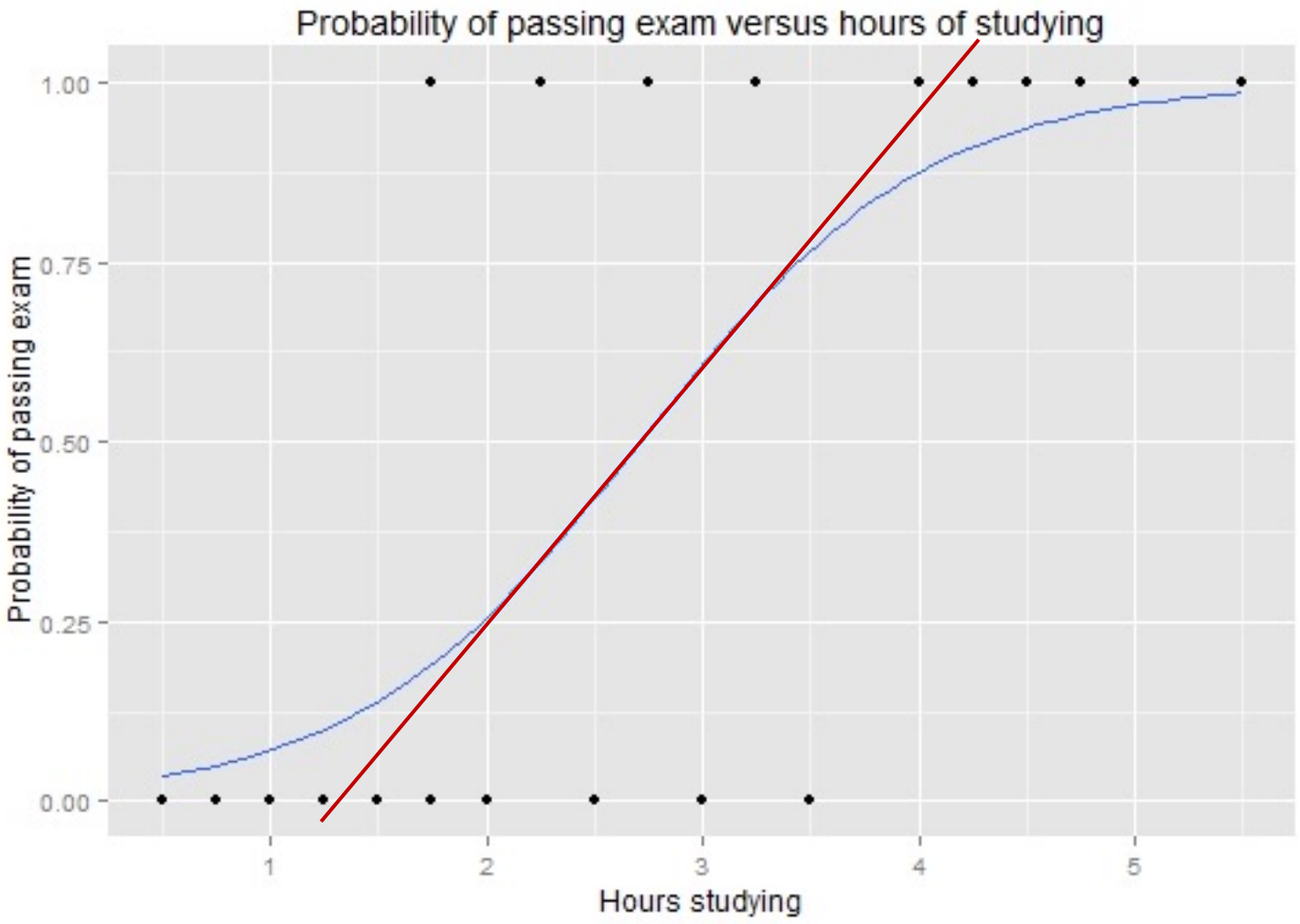


Image from Wikipedia

Logit function (log-odd)

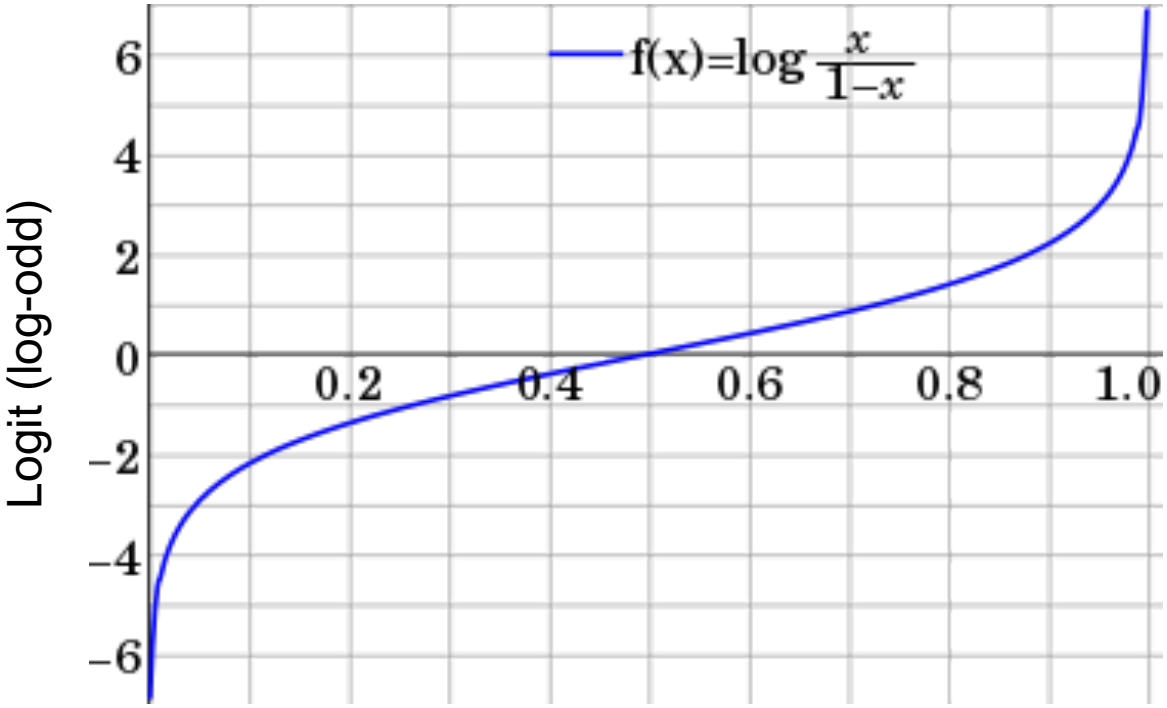


Image from Wikipedia

- The range of linear regression is $(-\infty, \infty)$
- The range of probability is $[0, 1]$
- Can we map $\mathrm{p} \in [0, 1]$ to $(-\infty, \infty)$?
  - Yes, with $r = \log\left(\frac{p}{1-p}\right)$

Logistic regression

- Model
  - $\log\left(\frac{\hat{y}^{(i)}}{1-\hat{y}^{(i)}}\right) = \text{logit}\left(\hat{y}^{(i)}\right) = f\left(x^{(i)}\right) = b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)}$
- Can we derive $\hat{y}^{(i)}$?
  - $\frac{\hat{y}^{(i)}}{1-\hat{y}^{(i)}} = e^{b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)}}$
  - $\frac{1}{\hat{y}^{(i)}} - 1 = \frac{1-\hat{y}^{(i)}}{\hat{y}^{(i)}} = \frac{1}{e^{b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)}}}$
  - $\hat{y}^{(i)} = \frac{e^{b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)}}}{1 + e^{b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)}}}$
  - What happens when $f\left(x^{(i)}\right)$ approaches $-\infty$ or ∞?
- Loss function
  - MSE for logistic regression: Brier score $= \frac{1}{N}\sum_i \left(y_i - \hat{y}^{(i)}\right)^2$
  - But this is not being interpreted as probability

Conditional maximum likelihood

- Likelihood: $\mathrm{P}(y_i \mid x^{(i)}) = \hat{y}^{(i)^{y_i}} \left(1 - \hat{y}^{(i)}\right)^{1-y_i}$

- Maximum likelihood principle
  - We want to maximize the probability of observing y_i given $x^{(i)}$
  - Maximizing f(x) is equivalent to maximizing log[f(x)]

- Logistic regression objective
  - $\log[\mathrm{P}(y_i \mid x^{(i)})] = y_i \log(\hat{y}^{(i)}) + (1 - y_i)\log(1 - \hat{y}^{(i)})$
  - $\hat{y}^{(i)} = \dfrac{e^{b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)}}}{1 + e^{b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)}}}$

- Calculate the gradients using chain rules
- This is also called **cross-entropy**

Multiclass with logistic model

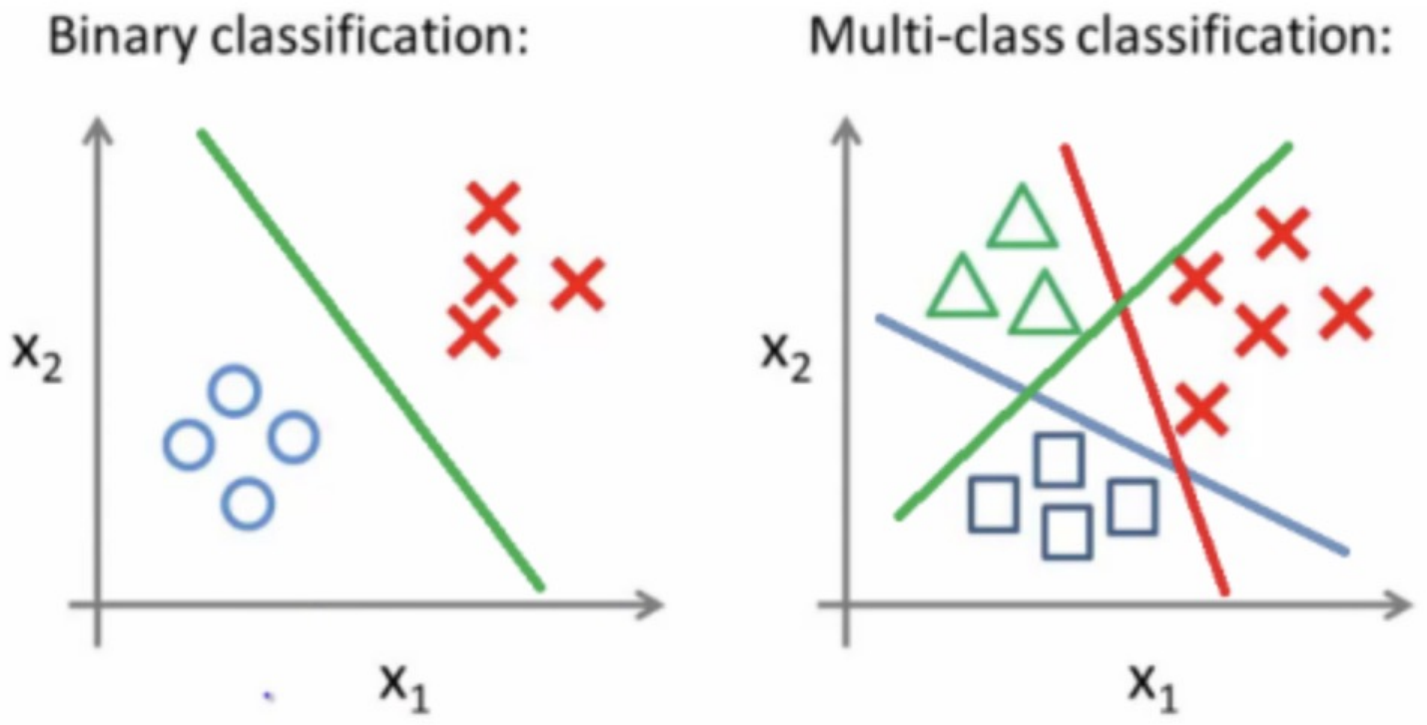


Image from medium.com

- One-versus-all
- Pairwise

Regularization

Regularization

- In addition to minimizing loss function (or maximizing likelihood), we can force the model to optimize other objectives
  - Make the magnitude of $b = (b_0, b_1, \ldots, b_n)$ small
  - Assign similar b_i's to clinical features from the same group

- Defining the L^k-norm of vector
  - L^1-norm $= \sum_i |b_i|$ = Manhattan distance to origin
  - L^2-norm $= \sqrt{\sum_i {b_i}^2}$ = Euclidean distance to origin

Ridge regression

$$\text{Loss} = C \cdot \sum_i \left(y_i - [b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)}] \right)^2 + \sum_i {b_i}^2$$

- Constant C control the relative importance of MSE and L^2

- Gradients
  - $\frac{\delta RIDGE}{\delta b_o} = \sum_i -2C \left(y_i - \left[b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)} \right] \right) + 2b_o$
  - $\frac{\delta RIDGE}{\delta b_1} = \sum_i -2C x_1^{(i)} \left(y_i - \left[b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)} \right] \right) + 2b_1$
  - $\cdots$
  - $\frac{\delta RIDGE}{\delta b_n} = \sum_i -2C x_n^{(i)} \left(y_i - \left[b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)} \right] \right) + 2b_n$

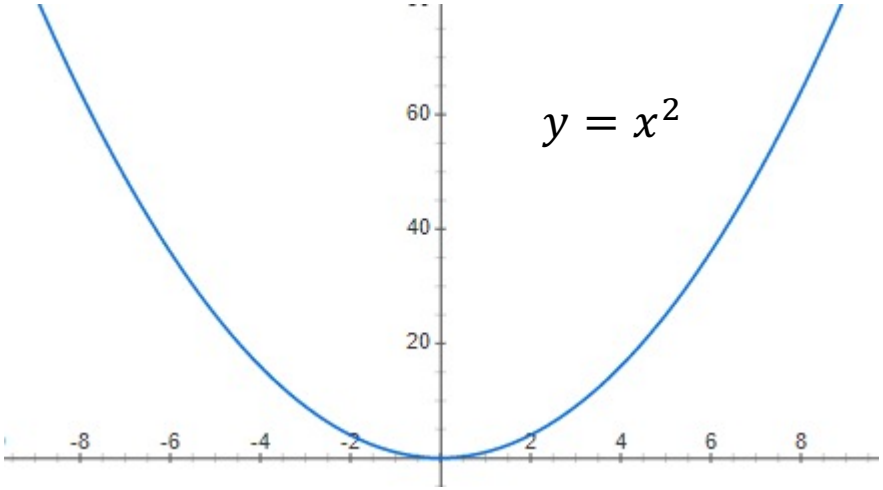
LASSO regression

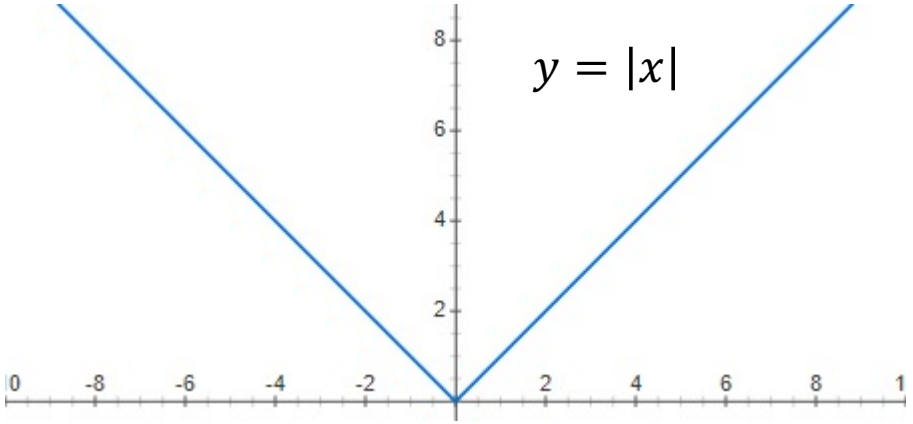
$$\text{Loss} = C \cdot \sum_i \left(y_i - [b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)}] \right)^2 + \sum_i |b_i|$$

- Least Absolute Shrinkage and **Selection Operator**
- Constant C control the relative importance of MSE and L^1

- Gradients
  - $\frac{\delta LASSO}{\delta b_o} = \sum_i -2C \left(y_i - \left[b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)} \right] \right) + \text{slope of |x| at } b_o$
  - $\frac{\delta LASSO}{\delta b_1} = \sum_i -2C x_1^{(i)} \left(y_i - \left[b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)} \right] \right) + \text{slope of |x| at } b_1$
  - …
  - $\frac{\delta LASSO}{\delta b_n} = \sum_i -2C x_n^{(i)} \left(y_i - \left[b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)} \right] \right) + \text{slope of |x| at } b_n$
  - **Note**: Slope of |x| is either +1 or -1

Impact of ridge vs Lasso





- Slope of parabola (=2x) gets smaller as x → 0
  - Larger step of x toward 0 is needed to reduce the value of y as x → 0
  - L2-regularization reduces coefficient's magnitude but rarely push it all the way to zero
- Slope of |x| is always ±1
  - A unit step in x toward 0 always results in unit reduction in y
  - L1-regularization pushes unimportant coefficients to zero

Ridge regression in Python

sklearn.linear_model.Ridge

class `sklearn.linear_model.`**`Ridge`**(*alpha=1.0, *, fit_intercept=True, normalize=False, copy_X=True, max_iter=None, tol=0.001, solver='auto', random_state=None*) [source]

Linear least squares with l2 regularization.

Minimizes the objective function:

```
||y - Xw||^2_2 + alpha * ||w||^2_2
```

This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. Also known as Ridge Regression or Tikhonov regularization. This estimator has built-in support for multi-variate regression (i.e., when y is a 2d-array of shape (n_samples, n_targets)).

Read more in the User Guide.

Parameters:

alpha : *{float, ndarray of shape (n_targets,)}, default=1.0*

Regularization strength; must be a positive float. Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization. Alpha corresponds to `1 / (2C)` in other linear models such as `LogisticRegression` or `LinearSVC`. If an array is passed, penalties are assumed to be specific to the targets. Hence they must correspond in number.

Lasso regression in Python

sklearn.linear_model.Lasso

class `sklearn.linear_model.` **`Lasso`**(*alpha=1.0, *, fit_intercept=True, normalize=False, precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic'*) [source]

Linear Model trained with L1 prior as regularizer (aka the Lasso)

The optimization objective for Lasso is:

```
(1 / (2 * n_samples)) * ||y - Xw||^2_2 + alpha * ||w||_1
```

Technically the Lasso model is optimizing the same objective function as the Elastic Net with `l1_ratio=1.0` (no L2 penalty).

Read more in the User Guide.

Parameters:

alpha : *float, default=1.0*

Constant that multiplies the L1 term. Defaults to 1.0. `alpha = 0` is equivalent to an ordinary least square, solved by the `LinearRegression` object. For numerical reasons, using `alpha = 0` with the `Lasso` object is not advised. Given this, you should use the `LinearRegression` object.

Regularized logistic regression

- Similar to Ridge and LASSO linear regressions
  - Ridge logistic: $C \cdot \log[\mathrm{P}(y_i \mid x^{(i)})] + \sum_i {b_i}^2$
  - LASSO logistic: $C \cdot \log[\mathrm{P}(y_i \mid x^{(i)})] + \sum_i |b_i|$

sklearn.linear_model.LogisticRegression

class `sklearn.linear_model.`**`LogisticRegression`**(*penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*) [source]

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. **Note that regularization is applied by default**. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization with primal formulation, or no regularization. The 'liblinear' solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty. The Elastic-Net regularization is only supported by the 'saga' solver.

ElasticNet = Ridge + LASSO

$$\text{Loss} = \sum_i \left(y_i - [b_0 + b_1 x_1^{(i)} + \cdots + b_n x_n^{(i)}] \right)^2 + \alpha \cdot \sum_i |b_i| + c\alpha \cdot \sum_i {b_i}^2$$

- Add both L^1-norm and L^2-norm with different weights
- Note that some formulation of regularization asks user to provide the weight of MSE term, *C*, and some asks for the weight of the regularization term, α
- To find *C* or α that yields the best model, we often search in the range of 0.001, 0.01, 0.1, 1.0, 10.0, 100.0

ElasticNet in Python

sklearn.linear_model.ElasticNet

class sklearn.linear_model.ElasticNet(*alpha=1.0, *, l1_ratio=0.5, fit_intercept=True, normalize='deprecated', precompute=False, max_iter=1000, copy_X=True, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic'*) [source]

Linear regression with combined L1 and L2 priors as regularizer.

Minimizes the objective function:

```
1 / (2 * n_samples) * ||y - Xw||^2_2
+ alpha * l1_ratio * ||w||_1
+ 0.5 * alpha * (1 - l1_ratio) * ||w||^2_2
```

If you are interested in controlling the L1 and L2 penalty separately, keep in mind that this is equivalent to:

```
a * ||w||_1 + 0.5 * b * ||w||_2^2
```

where:

```
alpha = a + b and l1_ratio = a / (a + b)
```

Support Vector Machine (SVM)

Separating hyperplane

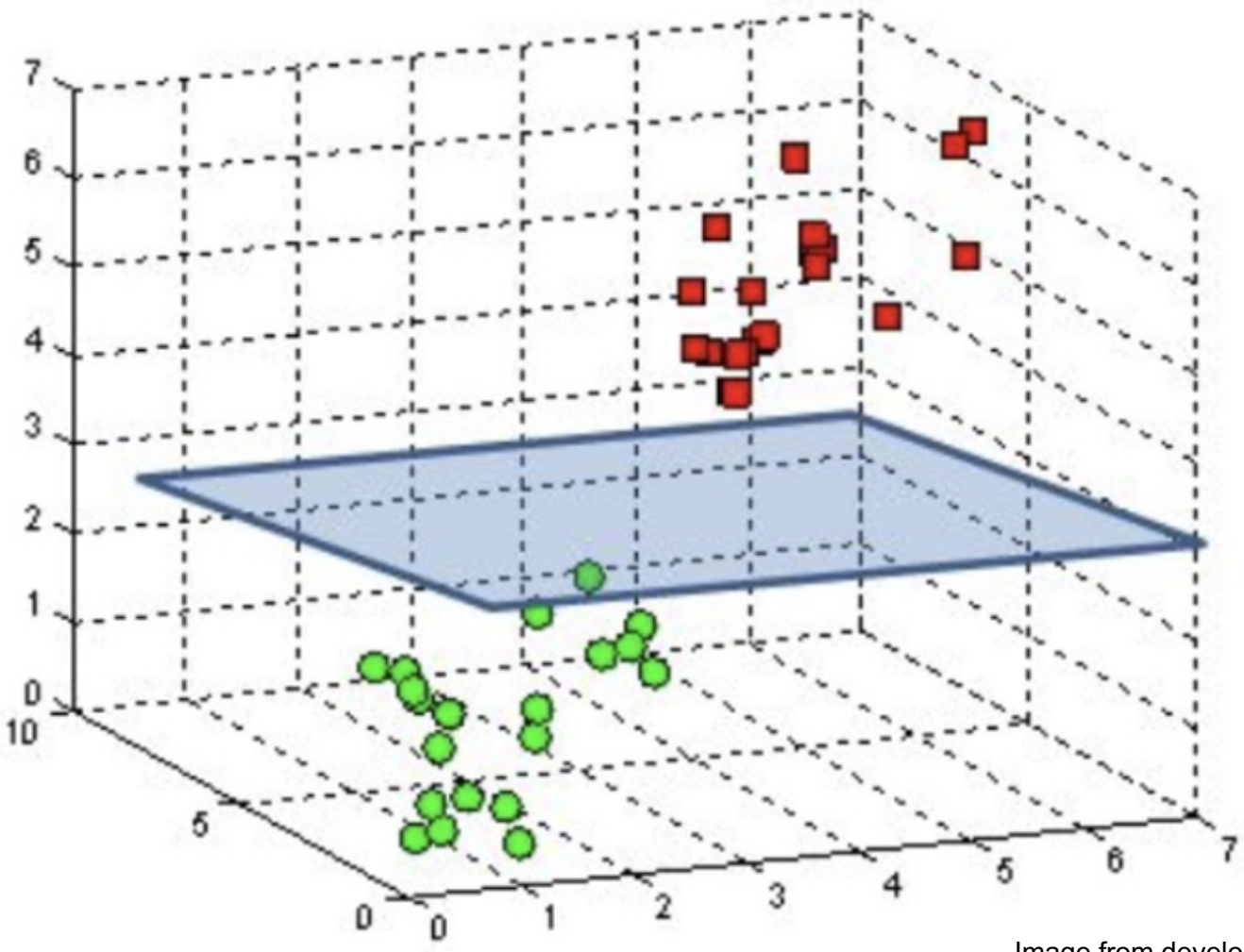
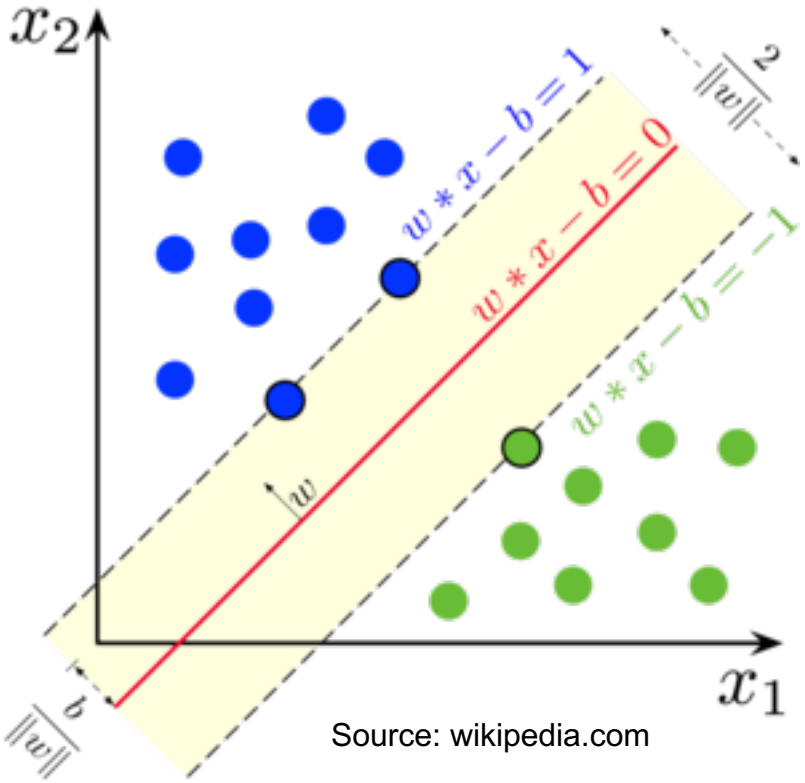


Image from developer.xilinx.com

- *n-1*-dim hyperplane divides *n*-dim space into two parts

Separating hyperplane and margin



Source: wikipedia.com

Hyperplane equation

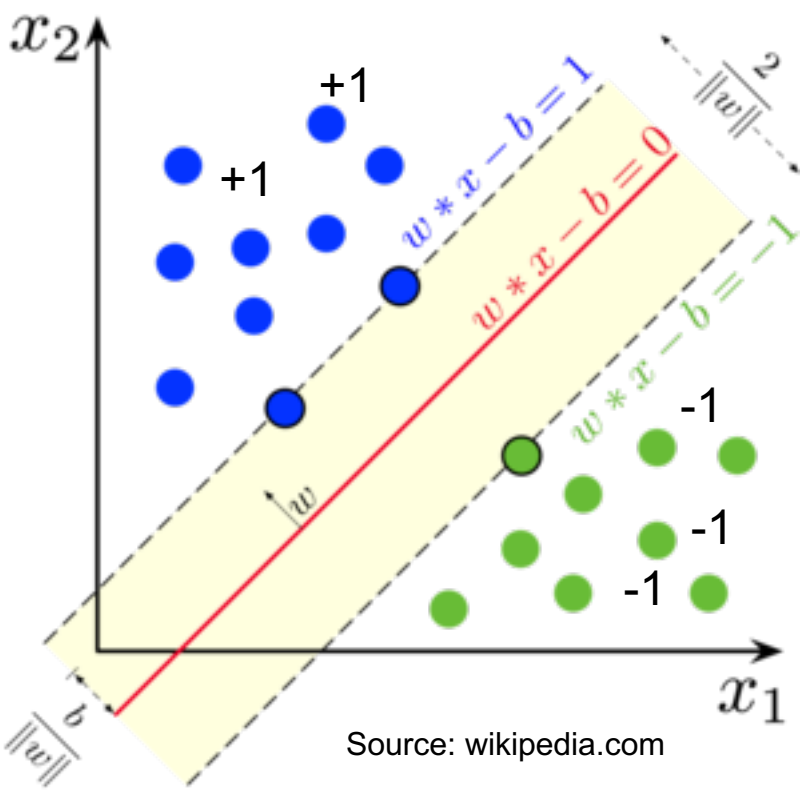
$$w_1 x_1 + \cdots + w_n x_n - b = 0$$

Scale the space so that the nearest data points on each side of the hyperplane satisfies

$$w_1 x_1 + \cdots + w_n x_n - b = \pm 1$$

Then, margin $= \frac{2}{\|w\|_2}$ where $\|w\|_2$ is the L-2 norm of *w*

SVM as an optimization problem



Source: wikipedia.com

Data points:

$$\{x_i \mid i = 1, \ldots, n\}$$

Labels:

$$y_i \in \{+1, -1\}$$

Hyperplane:

$$w_1 x_1 + \cdots + w_n x_n - b = 0$$

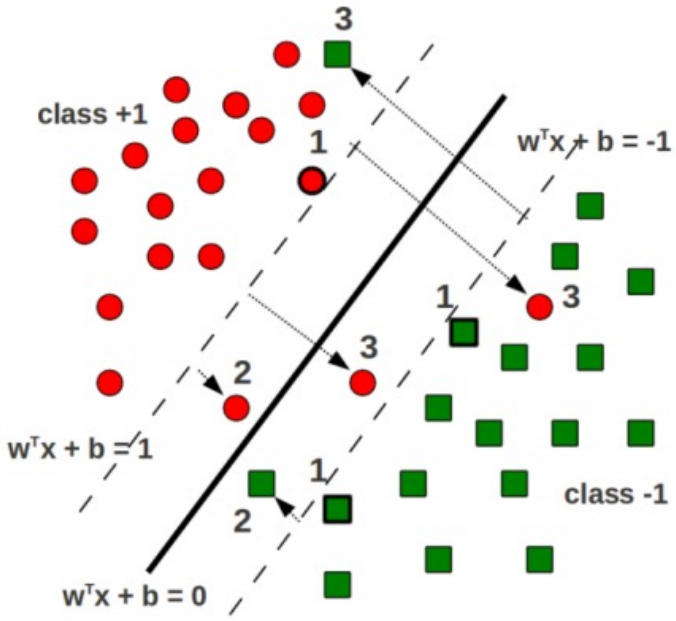
Optimization problem:

Minimize $\|w\|_2$ subject to

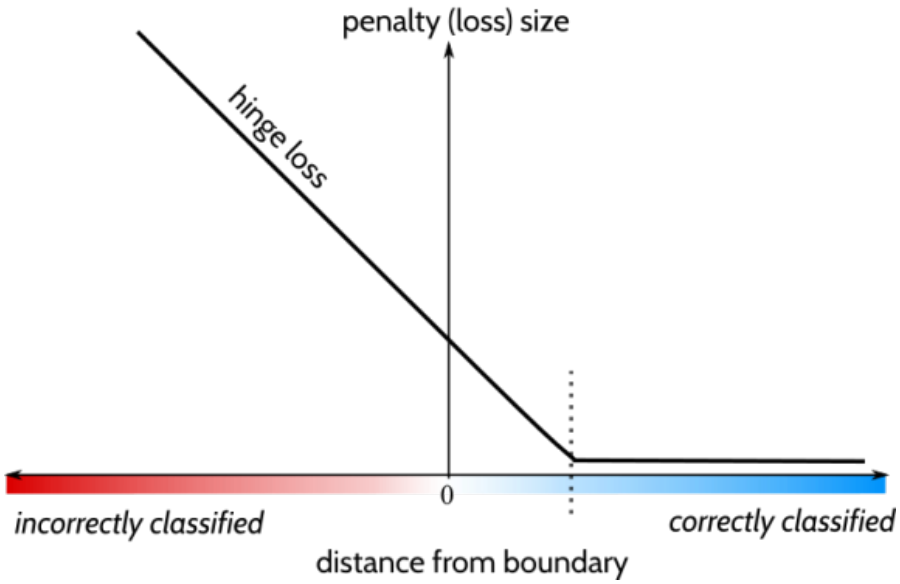
$y_i(w \cdot x_i - b) \geq 1$ for every i

- By construction, $w \cdot x_i - b$ is either ≥ 1 or ≤ -1
- Multiplication by y_i flips the sign for the negative class

But most data are not linearly separable!



Source: cs.umd.edu/cmsc422 class



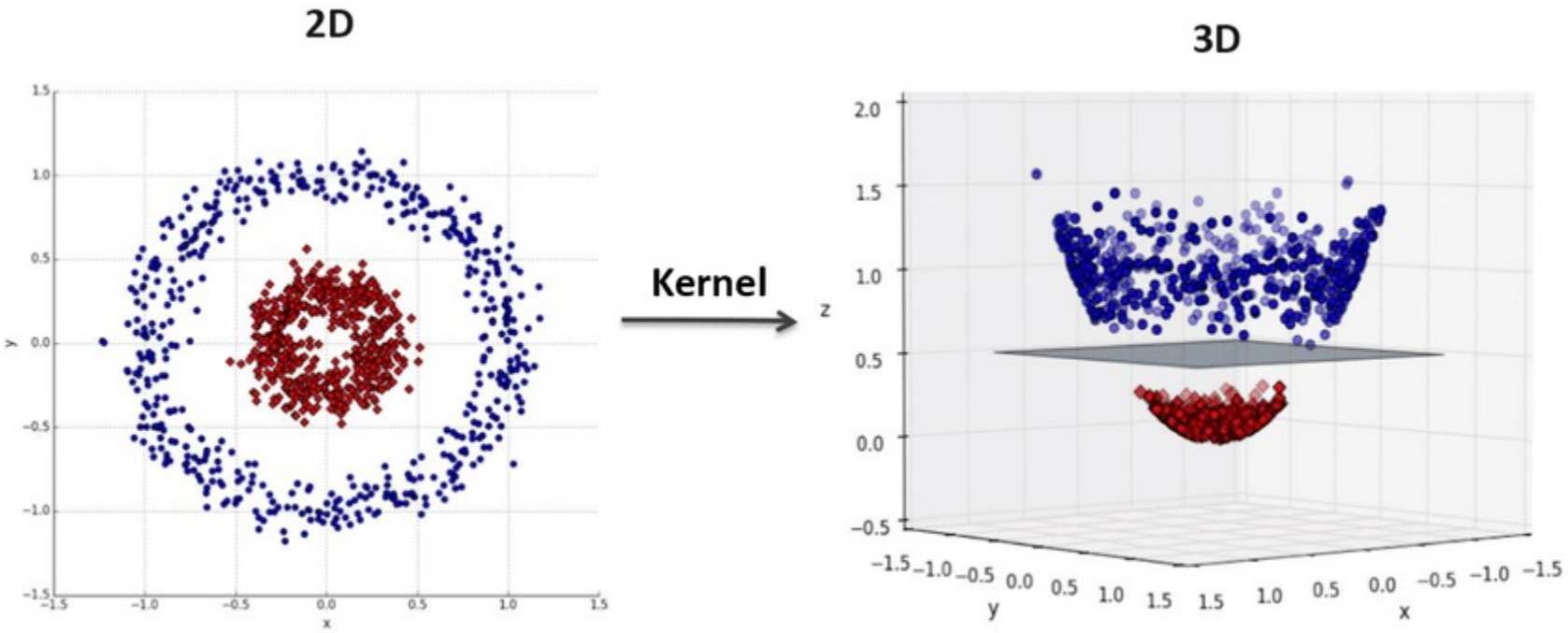
Source: towarddatasciences.com

- **Hinge loss**: $\max(0, 1 - y_i(w \cdot x_i - b))$
  - Penalize misclassification and data points lying within the margin

- Balance error and margin
  - Loss = $C \cdot \sum \max(0, 1 - y_i(w \cdot x_i - b)) + \frac{1}{2}\|w\|^2$

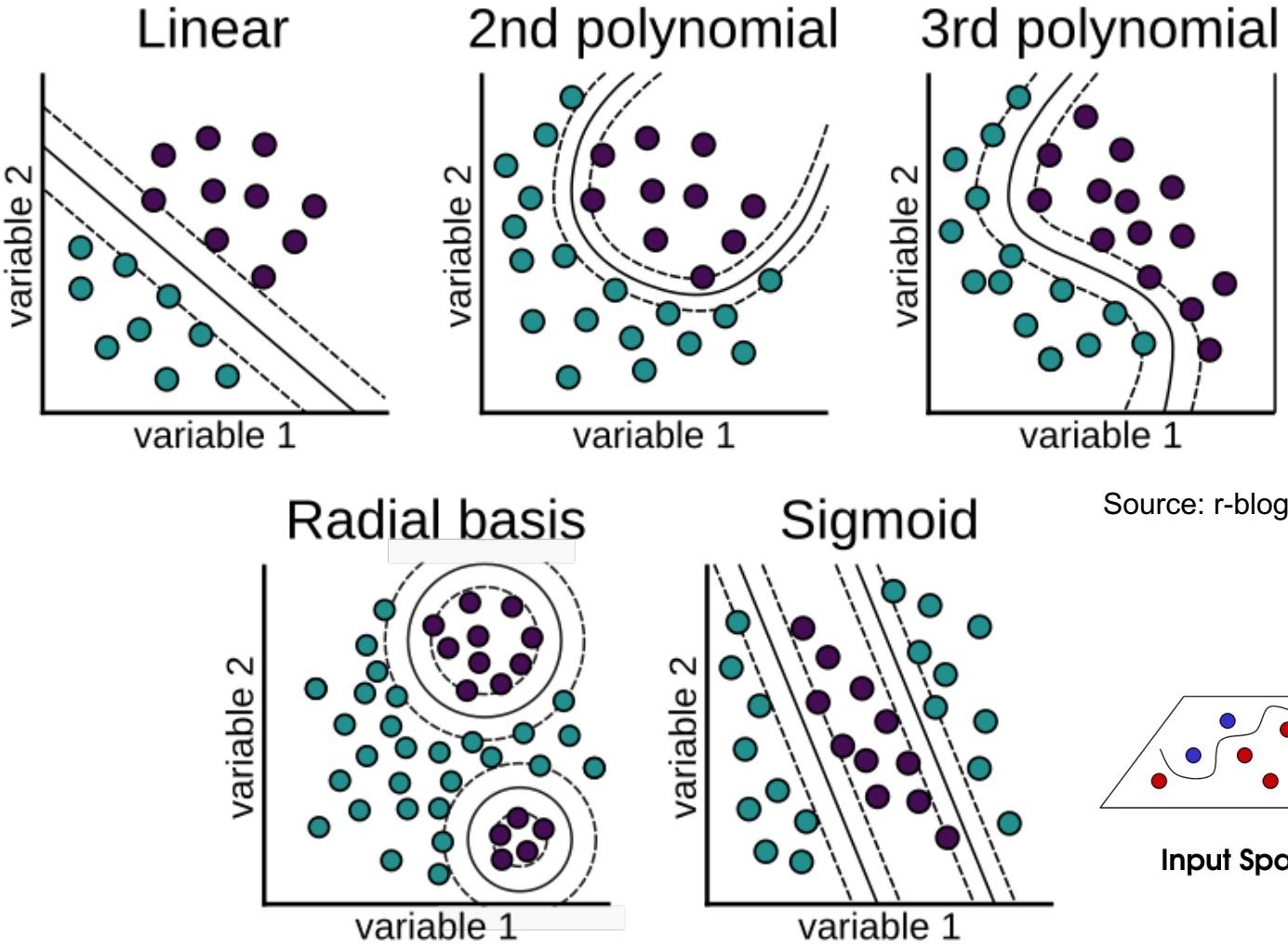
SVM for non-linear data

Feature transformation (kernel)



- Transform (x, y) to (x, y, $x^2 + y^2$)
  - Blue data points are further away from origin → larger $x^2 + y^2$
  - Separating hyperplane is a linear combination of x, y, and $x^2 + y^2$ which is nonlinear with respect to x and y

Common SVM kernels



Linear SVM in Python

sklearn.svm.LinearSVC

class sklearn.svm.**LinearSVC**(*penalty='l2', loss='squared_hinge', *, dual=True, tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000*) [source]

Linear Support Vector Classification.

Similar to SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

This class supports both dense and sparse input and the multiclass support is handled according to a one-vs-the-rest scheme.

Read more in the User Guide.

Parameters:

penalty : *{'l1', 'l2'}, default='l2'*
Specifies the norm used in the penalization. The 'l2' penalty is the standard used in SVC. The 'l1' leads to `coef_` vectors that are sparse.

loss : *{'hinge', 'squared_hinge'}, default='squared_hinge'*
Specifies the loss function. 'hinge' is the standard SVM loss (used e.g. by the SVC class) while 'squared_hinge' is the square of the hinge loss. The combination of `penalty='l1'` and `loss='hinge'` is not supported.

dual : *bool, default=True*
Select the algorithm to either solve the dual or primal optimization problem. Prefer dual=False when n_samples > n_features.

tol : *float, default=1e-4*
Tolerance for stopping criteria.

C : *float, default=1.0*
Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive.

Nonlinear SVM in Python

sklearn.svm.SVC

class sklearn.svm.**SVC**(*, *C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using `LinearSVC` or `SGDClassifier` instead, possibly after a `Nystroem` transformer.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how `gamma`, `coef0` and `degree` affect each other, see the corresponding section in the narrative documentation: Kernel functions.

Read more in the User Guide.

Parameters:

C : *float, default=1.0*
Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf'*
Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

degree : *int, default=3*
Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

gamma : *{'scale', 'auto'} or float, default='scale'*
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

Extending SVM to regression task

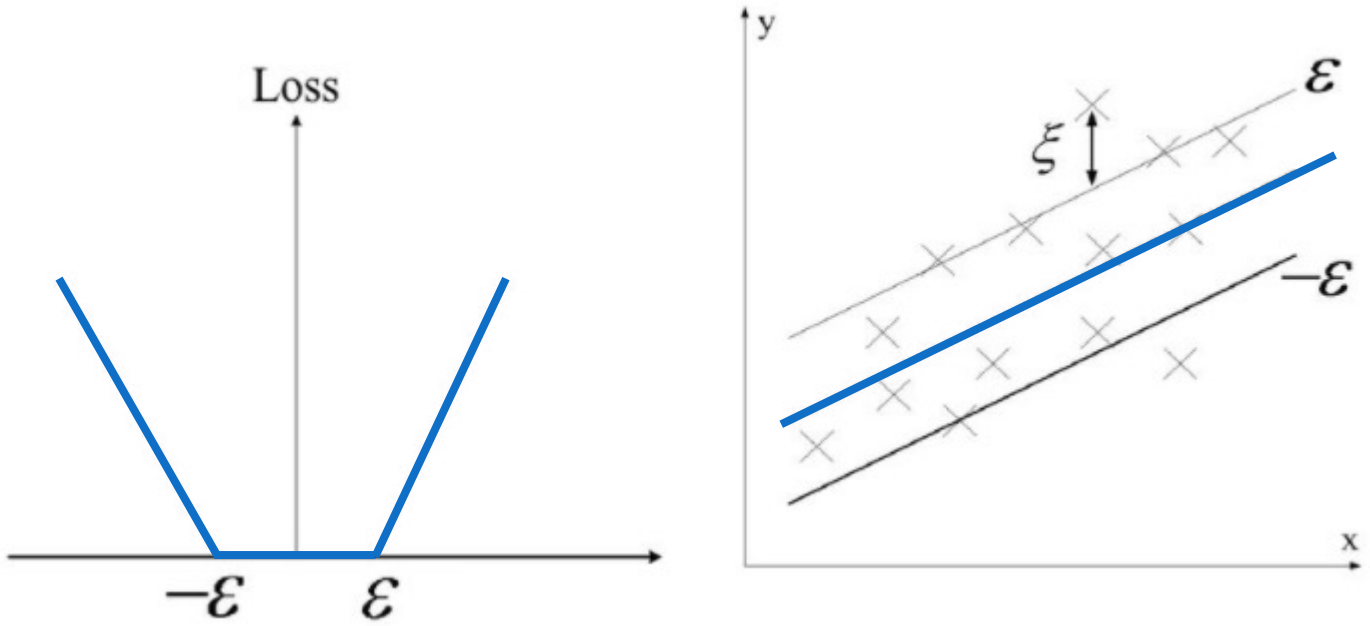


Image from https://slideplayer.com/slide/15044351/

- Penalize only data points with regression error $> \varepsilon$
- Loss = $\mathrm{C} \cdot \sum \max(0, |y_i(w \cdot x_i - b)| - \varepsilon) + \frac{1}{2}\|w\|^2$

Linear Support Vector Regression in Python

sklearn.svm.LinearSVR

class `sklearn.svm.LinearSVR`(*, *epsilon=0.0*, *tol=0.0001*, *C=1.0*, *loss='epsilon_insensitive'*, *fit_intercept=True*, *intercept_scaling=1.0*, *dual=True*, *verbose=0*, *random_state=None*, *max_iter=1000*) [source]

Linear Support Vector Regression.

Similar to SVR with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

This class supports both dense and sparse input.

Read more in the User Guide.

New in version 0.16.

Parameters:

epsilon : *float, default=0.0*
Epsilon parameter in the epsilon-insensitive loss function. Note that the value of this parameter depends on the scale of the target variable y. If unsure, set `epsilon=0`.

tol : *float, default=1e-4*
Tolerance for stopping criteria.

C : *float, default=1.0*
Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive.

loss : *{'epsilon_insensitive', 'squared_epsilon_insensitive'}, default='epsilon_insensitive'*
Specifies the loss function. The epsilon-insensitive loss (standard SVR) is the L1 loss, while the squared epsilon-insensitive loss ('squared_epsilon_insensitive') is the L2 loss.

Any question?