Data Analysis and Knowledge Discovery

Regression

Jukka Heikkonen

University of Turku
Department of Information Technology

Jukka.Heikkonen@utu.fi

Regression

- Given: *training set* of input-output pairs $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$
- Learn: model f such that given a new input $\mathbf{x}$ for which corresponding output y unknown
  - $f(\mathbf{x}) \approx y$
- $\mathbf{x} \in \mathbb{R}^d$ is a d-dimensional feature vector (input)
- $y \in \mathbb{R}$ is the real-valued output to be predicted

Linear regression

The linear model, a simple yet popular choice:

$$f(\mathbf{x}) = w_1 \cdot x_1 + ... + w_d \cdot x_d + b$$

- $x_1, ..., x_d$, feature values
- $w_1, ..., w_d$ model coefficients
- $b \in \mathbb{R}$ intercept term

Linear regression

In sum notation, this is written as

$$f(\mathbf{x}) = \sum_{i=1}^{d} w_i \cdot x_i + b$$

- $x_1, ..., x_d$, feature values
- $w_1, ..., w_d$ model coefficients
- $b \in \mathbb{R}$ intercept term

Linear regression

Append value 1 to to the beginning of each feature vector (new constant valued feature x_0), and define a new coefficient $w_0 = b$

$$f(\mathbf{x}) = \sum_{i=1}^{d} w_i \cdot x_i + b = \sum_{i=0}^{d} w_i \cdot x_i$$

- $x_0 = 1$ the constant valued feature
- $x_1, ..., x_d$, feature values
- $w_0, ..., w_d$ model coefficients

(A standard trick, re-naming the bias term 'b' just makes the following math and algorithmics a bit simpler.)

Linear regression

Finally, defining a coefficient vector $\mathbf{w} = [w_0, ..., w_d]$, we can reformulate this as the inner product between the model and feature vectors

$$f(\mathbf{x}) = \sum_{i=0}^{d} w_i \cdot x_i = \mathbf{w}^\mathsf{T}\mathbf{x}$$

- $\mathbf{x}$ feature vector
- $\mathbf{w}$ model vector

Regression line (single feature case)

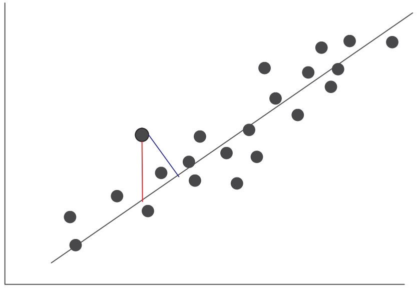
given: A data set for two continuous attributes x (input feature) and y (output).

It is assumed that there is an approximate linear dependency between x and y: $y \approx w_1 x + w_0$

Find a regression line (i.e. determine the parameters w_1 and w_0) such that the line fits the data as good as possible.

What is a good fit?

Regression



y-distance Euclidean distance

Usually, the mean square error in *y*-direction is chosen as error measure (to be minimized).
It is equivalent to minimize the sum of squared errors in *y*-direction.

Regression

Other reasonable error measures:

- mean absolute distance in y-direction
- mean Euclidean distance
- maximum absolute distance in y-direction (or equivalently: the maximum squared distance in y-direction)
- maximum Euclidean distance
- ...

Regression line

Given data $(\mathbf{x}_i, y_i)$ $(i = 1, \dots, n)$, the least squares error function is

$$\sum_{i=1}^{n} \left(\mathbf{w}^{\mathsf{T}} \mathbf{x}_i - y_i \right)^2$$

Least squares

Classical least-squares method (1809, Carl Friedrich Gauss):

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^{n} \left(\mathbf{w}^\mathsf{T} \mathbf{x}_i - y_i \right)^2$$

- choose model $\mathbf{w}^*$ having smallest least-squares error
- can sometimes work fine if dimensionality d much smaller than training set size n
- however, especially prone to overfitting in high dimensions (also, no unique solution if $d > n$)
- Sensitive to outliers

Regularized least-squares

Regularized least-squares, aka ridge regression:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \left(\mathbf{w}^\mathsf{T} \mathbf{x}_i - y_i \right)^2 + \lambda \sum_{i=0}^{d} w_i^2 \right\}$$

- regularization term penalizes too complex models
- $\lambda > 0$ regularization parameter (can be chosen with cross-validation)
- unique solution, much more robust than basic least-squares fitting, especially for high-dimensional data

Regularization for linear regression

Regularized least-squares regression:

$$\operatorname*{argmin}_{\mathbf{w}} \left\{ \underbrace{\sum_{i=1}^{n} \left(\mathbf{w}^{\mathsf{T}} \mathbf{x}_i - y_i \right)^2}_{\text{Training set error}} + \lambda \underbrace{\sum_{i=0}^{d} w_i^2}_{\text{Regularizer}} \right\}$$

Regularization for linear regression

Let us assume the following data structures:

- **X**: a nxd -sized data matrix, one row for each instance
- **y**: a n-length column vector of correct outputs, one element for each instance
- **w**: d-length column vector of coefficients, we wish to learn from data
- (Minor technical detail: if we use the intercept term in our model, **X** has one additional column of ones, and **w** correspondingly one more element. This can be useful especially on low-dimensional data, where the additional flexibility given to model may be helpful.)

Regularization for linear regression

Regularized least-squares regression:

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \underbrace{\sum_{i=1}^{n} \left(\mathbf{w}^{\mathsf{T}} \mathbf{x}_i - y_i \right)^2}_{\text{Training set error}} + \underbrace{\lambda \sum_{i=0}^{d} w_i^2}_{\text{Regularizer}} \right\}$$

The same in matrix form:

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \underbrace{(\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathsf{T}} (\mathbf{X}\mathbf{w} - \mathbf{y})}_{\text{Training set error}} + \underbrace{\lambda \mathbf{w}^{\mathsf{T}} \mathbf{w}}_{\text{Regularizer}} \right\}$$

Solving regularized least-squares

Objective function to be minimized

$$J(\mathbf{w}) = (\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathsf{T}}(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^{\mathsf{T}}\mathbf{w}$$

Gradient

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial}{\partial \mathbf{w}}\Big((\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathsf{T}}(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^{\mathsf{T}}\mathbf{w}\Big)$$

It can be shown that the global minimum of the objective function can be found at the point where the gradient is zero (due to convexity of the objective function).

Solving regularized least-squares

$$\begin{aligned}\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}}\Big((\mathbf{X}\mathbf{w}-\mathbf{y})^\mathsf{T}(\mathbf{X}\mathbf{w}-\mathbf{y}) + \lambda \mathbf{w}^\mathsf{T}\mathbf{w}\Big)\\ &= \frac{\partial}{\partial \mathbf{w}}\Big(\mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{w} - \mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{y} - \mathbf{y}^\mathsf{T}\mathbf{X}\mathbf{w} + \mathbf{y}^\mathsf{T}\mathbf{y} + \lambda \mathbf{w}^\mathsf{T}\mathbf{w}\Big)\end{aligned}$$

Recall: rules of matrix transposition

$$(\mathbf{M}\mathbf{N})^\mathsf{T} = \mathbf{N}^\mathsf{T}\mathbf{M}^\mathsf{T}$$

Solving regularized least-squares

$$\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}}\Big((\mathbf{X}\mathbf{w}-\mathbf{y})^{\mathsf{T}}(\mathbf{X}\mathbf{w}-\mathbf{y})+\lambda\mathbf{w}^{\mathsf{T}}\mathbf{w}\Big)\\
&= \frac{\partial}{\partial \mathbf{w}}\Big(\mathbf{w}^{\mathsf{T}}\mathbf{X}^{\mathsf{T}}\mathbf{X}\mathbf{w}-\mathbf{w}^{\mathsf{T}}\mathbf{X}^{\mathsf{T}}\mathbf{y}-\mathbf{y}^{\mathsf{T}}\mathbf{X}\mathbf{w}+\mathbf{y}^{\mathsf{T}}\mathbf{y}+\lambda\mathbf{w}^{\mathsf{T}}\mathbf{w}\Big)\\
&= \frac{\partial}{\partial \mathbf{w}}\Big(\mathbf{w}^{\mathsf{T}}(\mathbf{X}^{\mathsf{T}}\mathbf{X}+\lambda\mathbf{I})\mathbf{w}-2\mathbf{y}^{\mathsf{T}}\mathbf{X}\mathbf{w}+\mathbf{y}^{\mathsf{T}}\mathbf{y}\Big)
\end{aligned}$$

Solving regularized least-squares

$$\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}}\Big((\mathbf{X}\mathbf{w}-\mathbf{y})^\mathsf{T}(\mathbf{X}\mathbf{w}-\mathbf{y})+\lambda\mathbf{w}^\mathsf{T}\mathbf{w}\Big)\\
&= \frac{\partial}{\partial \mathbf{w}}\Big(\mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{w}-\mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{y}-\mathbf{y}^\mathsf{T}\mathbf{X}\mathbf{w}+\mathbf{y}^\mathsf{T}\mathbf{y}+\lambda\mathbf{w}^\mathsf{T}\mathbf{w}\Big)\\
&= \frac{\partial}{\partial \mathbf{w}}\Big(\mathbf{w}^\mathsf{T}(\mathbf{X}^\mathsf{T}\mathbf{X}+\lambda\mathbf{I})\mathbf{w}-2\mathbf{y}^\mathsf{T}\mathbf{X}\mathbf{w}+\mathbf{y}^\mathsf{T}\mathbf{y}\Big)\\
&= (\mathbf{X}^\mathsf{T}\mathbf{X}+\lambda\mathbf{I})\mathbf{w}+(\mathbf{X}^\mathsf{T}\mathbf{X}+\lambda\mathbf{I})^\mathsf{T}\mathbf{w}-2\mathbf{X}^\mathsf{T}\mathbf{y}
\end{aligned}$$

Gradient rules

$$\frac{\partial}{\partial \mathbf{w}}\mathbf{v}^\mathsf{T}\mathbf{w}=\mathbf{v}$$

$$\frac{\partial}{\partial \mathbf{w}}\mathbf{w}^\mathsf{T}\mathbf{M}\mathbf{w}=\mathbf{M}\mathbf{w}+\mathbf{M}^\mathsf{T}\mathbf{w}$$

Solving regularized least-squares

$$\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}}\Big((\mathbf{X}\mathbf{w}-\mathbf{y})^\mathsf{T}(\mathbf{X}\mathbf{w}-\mathbf{y})+\lambda\mathbf{w}^\mathsf{T}\mathbf{w}\Big)\\
&= \frac{\partial}{\partial \mathbf{w}}\Big(\mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{w}-\mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{y}-\mathbf{y}^\mathsf{T}\mathbf{X}\mathbf{w}+\mathbf{y}^\mathsf{T}\mathbf{y}+\lambda\mathbf{w}^\mathsf{T}\mathbf{w}\Big)\\
&= \frac{\partial}{\partial \mathbf{w}}\Big(\mathbf{w}^\mathsf{T}(\mathbf{X}^\mathsf{T}\mathbf{X}+\lambda\mathbf{I})\mathbf{w}-2\mathbf{y}^\mathsf{T}\mathbf{X}\mathbf{w}+\mathbf{y}^\mathsf{T}\mathbf{y}\Big)\\
&= (\mathbf{X}^\mathsf{T}\mathbf{X}+\lambda\mathbf{I})\mathbf{w}+(\mathbf{X}^\mathsf{T}\mathbf{X}+\lambda\mathbf{I})^\mathsf{T}\mathbf{w}-2\mathbf{X}^\mathsf{T}\mathbf{y}\\
&= 2(\mathbf{X}^\mathsf{T}\mathbf{X}+\lambda\mathbf{I})\mathbf{w}-2\mathbf{X}^\mathsf{T}\mathbf{y}
\end{aligned}$$

Solving regularized least-squares

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = 2(\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda \mathbf{I})\mathbf{w} - 2\mathbf{X}^\mathsf{T}\mathbf{y} = 0$$

$$(\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda \mathbf{I})\mathbf{w} = \mathbf{X}^\mathsf{T}\mathbf{y}$$

Solving regularized least-squares

$$(\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda\mathbf{I})\mathbf{w} = \mathbf{X}^\mathsf{T}\mathbf{y}$$

Regularized least-squares

- Given: data matrix $\mathbf{X}$, outputs $\mathbf{y}$, hyperparameter λ
- Solve the above linear system to find model coefficients $\mathbf{w}$
- Can be solved using any standard linear algebra package (e.g. numpy.linalg.solve( ))
- Computational complexity: $O(d^3 + d^2 n)$, memory usage $O(d^2 + dn)$
- feasible if dimensionality d is not too large (at most couple of thousands), but what if $d >> n$?

Code for training regularized least-squares

```
import numpy as np

def rls(X, y, regparam):
    #X: nxd data matrix
    #y: n-length vector of outputs
    #regparam > 0: parameter
    #returns: coefficients w
    d = X.shape[1]
    I = np.eye(d)
    A = np.dot(X.T, X) + regparam*I
    b = np.dot(X.T, y)
    w = np.linalg.solve(A, b)
    return w
```

Solving regularized least-squares, the other way

$$\begin{aligned}(\mathbf{X}\mathbf{X}^{\mathsf{T}} + \lambda\mathbf{I})\mathbf{a} &= \mathbf{y} \\ \mathbf{w} &= \mathbf{X}^{\mathsf{T}}\mathbf{a}\end{aligned}$$

Regularized least-squares

- It can be shown, that regularized least-squares can equivalently be trained by solving a nxn-sized linear system of equations
- Much more efficient than previous form, if dimensionality much larger than sample size (e.g. microarray data in bioinformatics)
- Computational complexity: $O(n^3 + n^2 d)$, memory usage $O(n^2 + dn)$

Code for training regularized least-squares, dual form

```
import numpy as np

def rls(X, y, regparam):
    #X: nxd data matrix
    #y: n-length vector of outputs
    #regparam > 0: parameter
    #returns: coefficients w
    n = X.shape[0]
    I = np.eye(n)
    A = np.dot(X, X.T) + regparam*I
    a = np.linalg.solve(A, y)
    w = np.dot(X.T, a)
    return w
```

Code for lazy people

```
from sklearn.linear_model import Ridge

def rls_using_sklearn(X, y, regparam):
    learner = Ridge(alpha=regparam,
    fit_intercept=False)
    learner.fit(X, y)
   w = learner.coef_
    return w
```

Should give same results as previous codes, the underlying implementation automatically decides, which of the previously shown formulations is solved depending on the values of d and n. Minor technical detail: Usually you should set fit_intercept=True, this would be equivalent to appending a constant feature to each feature vector in the previous examples.

Applying the model on new data

```
import numpy as np

def predict(x_test, w):
    #x_test: test instance, vector of d-features
    #w: vector of d-coefficients
    return np.dot(x_test, w)
```

What did we learn?

- linear model for regression
- regularized least squares, aka ridge regression, aka least-squares support vector machine
- idea: minimize mean squared error on training data, use a regularization term to penalize model complexity (here coefficients squared)
- unique optimal solution by solving a linear system of equations
- efficient to train, either by solving *dxd* or *nxn* -sized linear system (choose minimum)
- produces a compact linear model, that can be very efficiently be used to predict on test instances
- often gives good predictive performance, assuming the problem is not highly non-linear (you could also try, say, k-nearest neighbour, in case it is)

About model selection

- challenge: need to do model selection (default choices like $\lambda = 1$ may often work suboptimally or not at all)
- regularization parameter λ, value needs to be chosen for example by 10-fold or leave-one-out cross-validation
  - Advanced topic: very fast cross-validation algorithms exist for regularized least-squares (especially fast leave-one-out widely known and found in most decent implementations)
  - My rule of thumb: in most cases selecting λ by choosing the parameter leading to lowest cross-validation error from the exponential grid $\{2^{-15}, ..., 2^{15}\}$ should suffice.

Linear regression, other methods

Wait, what if we had chosen to minimize some other reasonable criterion?

- for example, what if we would measure the fit of model to training data with absolute error $|f(\mathbf{x}) - y|$ instead of squared error $(f(\mathbf{x}) - y)^2$?
- or maybe I could penalize the model coefficients instead of the squared terms w_i^2 with, say, absolute magnitudes $|w_i|$?
- or maybe...

Linear regression, other methods

Regularized least-squares regression:

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \underbrace{\sum_{i=1}^{n} \left(\mathbf{w}^{\mathsf{T}} \mathbf{x}_i - y_i \right)^2}_{\text{Training set error}} + \lambda \underbrace{\sum_{i=0}^{d} w_i^2}_{\text{Regularizer}} \right\}$$

Lasso:

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \underbrace{\sum_{i=1}^{n} \left(\mathbf{w}^{\mathsf{T}} \mathbf{x}_i - y_i \right)^2}_{\text{Training set error}} + \lambda \underbrace{\sum_{i=0}^{d} |w_i|}_{\text{Regularizer}} \right\}$$

Linear regression, other methods

Regularized least-squares regression:

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \underbrace{\sum_{i=1}^{n} \left(\mathbf{w}^{\mathsf{T}} \mathbf{x}_i - y_i \right)^2}_{\text{Training set error}} + \lambda \underbrace{\sum_{i=0}^{d} w_i^2}_{\text{Regularizer}} \right\}$$

Elastic Net:

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \underbrace{\sum_{i=1}^{n} \left(\mathbf{w}^{\mathsf{T}} \mathbf{x}_i - y_i \right)^2}_{\text{Training set error}} + \underbrace{\lambda_1 \sum_{i=0}^{d} |w_i| + \lambda_2 \sum_{i=0}^{d} w_i^2}_{\text{Regularizer}} \right\}$$

Linear regression, other methods

Regularized least-squares regression:

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \underbrace{\sum_{i=1}^{n} \left(\mathbf{w}^{\mathsf{T}} \mathbf{x}_i - y_i \right)^2}_{\text{Training set error}} + \lambda \underbrace{\sum_{i=0}^{d} w_i^2}_{\text{Regularizer}} \right\}$$

Support vector regression

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \underbrace{\sum_{i=1}^{n} max \left(0, |\mathbf{w}^{\mathsf{T}} \mathbf{x}_i - y_i| - \epsilon \right)}_{\text{Training set error}} + \lambda \underbrace{\sum_{i=0}^{d} w_i^2}_{\text{Regularizer}} \right\}$$

Linear regression, other methods

- Why you might need to consider these alternatives
  - Lasso: leads to sparse solutions with many zero-coefficients, good for feature selection (sklearn.linear_model.Lasso)
  - Elastic Net: tries to get best of both worlds by interpolating between basic regularized least-squares and Lasso (sklearn.linear_model.ElasticNet)
  - Support vector regression: more robust towards outliers in data (sklearn.svm.SVR)
- Then again...
  - much more difficult to optimize, there is no analytical solution for minimizer, and since these are non-smooth basic gradient descent methods will not work well
  - in practice, often will not yield much better predictive accuracy
- Huge number of other approaches also out there

Classification as regression

- Given: *training set* of input-output pairs $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$
- Learn: model f such that given a new input $\mathbf{x}$ for which corresponding output y unknown
  - $f(\mathbf{x}) \approx y$
- $\mathbf{x} \in \mathbb{R}^d$ is a d-dimensional feature vector (input)
- $y \in 0, 1$ denotes, whether instance belongs to class 0 or 1

Classification as regression

A two-class classification problem (with classes encoded as -1 and 1) can be viewed as regression problem.

The regression function will usually not yield exact outputs -1 and 1, but the classification decision can be made by considering 0 as a cut-off value.

Problem: The objective functions aims at minimizing the function approximation error (for example, the mean squared error), but not misclassifications.

Classification as regression

For multiclass problems we may try to learn a single regression function without enumerating the classes:

- This leads to interpolation errors. For example: A data object between class 1 and 3 might be classified as class 2 by the regression function.

Another possibilty is to train a classifier (regression function) for each class against all other classes (one-versus-all). At prediction time all classifiers vote, assign to class with highest predicted value.

Probably the best approach is to have a single multiple output model (binary coding of classes) and select the class that has the highest value.

Regularized logistic regression

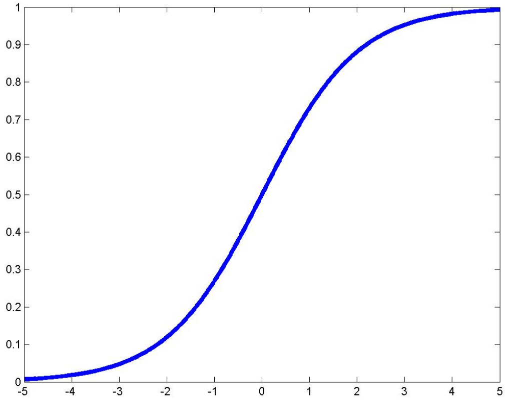
- **Given:** A set of data points $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ each of which belongs to one of the two classes denoted as -1 and 1.
- **Desired:** A simple description of the function $p(y = 1|\mathbf{x})$, probability that instance belongs to class 1 given features $\mathbf{x}$ (obviously, $p(y = -1|\mathbf{x}) = 1 - p(y = 1|\mathbf{x})$)
- **Approach:** Describe p by a logistic function:

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

- A linear model, with logistic function used to squeeze the predictions between 0 and 1. How to learn the coefficients from data?

Sigmoid function

$$\frac{1}{1 + e^{-x}}$$



Regularized logistic regression

Minimize the logarithm of the likelihood of training data, with regularization added

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \underbrace{\sum_{i=1}^{n} log \left(1 + e^{-y_i \mathbf{w}^{\mathsf{T}} \mathbf{x}_i}\right)}_{\text{Training set error}} + \lambda \underbrace{\sum_{i=0}^{d} w_i^2}_{\text{Regularizer}} \right\}$$

Regularized logistic regression

- logistic regression: a very classical linear model for classification
- predictions scaled between 0 and 1, can be interpreted as probabilities
- regularization can be used to help control model complexity
- may lead to better classification accuracy than using standard regularized least-squares regression
- can be trained with standard gradient descent optimization
- sklearn.linear_model.LogisticRegression

Non-linear models

- what if the model to be learned is highly non-linear, and nearest neighbour does not work well enough?
- kernel methods: generalize basic linear models to handling non-linear data
- classification: kernel logistic regression, support vector machines
- regression: kernel regularized least-squares
- more complicated models, same basic idea: balance training error with model complexity
- outside the scope of this course