# # Regularization

# ## Agenda:

#

# 1. Overfitting (review)

# 2. Overfitting with linear models

# 3. Regularization of linear models

# 4. Regularized regression in scikit-learn

# 5. Regularized classification in scikit-learn

# 6. Comparing regularized linear models with unregularized linear models

# ## Part 1: Overfitting (review)

#

# \*\*What is overfitting?\*\*

#

# - Building a model that matches the training data "too closely"

# - Learning from the noise in the data, rather than just the signal

#

# \*\*How does overfitting occur?\*\*

#

# - Evaluating a model by testing it on the same data that was used to train it

# - Creating a model that is "too complex"

#

# \*\*What is the impact of overfitting?\*\*

#

# - Model will do well on the training data, but won't generalize to out-of-sample data

# - Model will have low bias, but high variance

# ### Overfitting with KNN

#

# ![Overfitting with KNN](images/iris\_01nn\_map.png)

# ### Overfitting with polynomial regression

#

# ![Overfitting with polynomial regression](images/polynomial\_overfitting.png)

# ### Overfitting with decision trees

#

# ![Overfitting with decision trees](images/salary\_tree\_deep.png)

# ## Part 2: Overfitting with linear models

#

# \*\*What are the general characteristics of linear models?\*\*

#

# - Low model complexity

# - High bias, low variance

# - Does not tend to overfit

#

# Nevertheless, \*\*overfitting can still occur\*\* with linear models if you allow them to have \*\*high variance\*\*. Here are some common causes:

# ### Cause 1: Irrelevant features

#

# Linear models can overfit if you include "irrelevant features", meaning features that are unrelated to the response. Why?

#

# Because it will learn a coefficient for every feature you include in the model, regardless of whether that feature has the \*\*signal\*\* or the \*\*noise\*\*.

#

# This is especially a problem when \*\*p (number of features) is close to n (number of observations)\*\*, because that model will naturally have high variance.

# ### Cause 2: Correlated features

#

# Linear models can overfit if the included features are highly correlated with one another. Why?

#

# From the [scikit-learn documentation](http://scikit-learn.org/stable/modules/linear\_model.html#ordinary-least-squares):

#

# > "...coefficient estimates for Ordinary Least Squares rely on the independence of the model terms. When terms are correlated and the columns of the design matrix X have an approximate linear dependence, the design matrix becomes close to singular and as a result, the least-squares estimate becomes highly sensitive to random errors in the observed response, producing a large variance."

# ### Cause 3: Large coefficients

#

# Linear models can overfit if the coefficients (after feature standardization) are too large. Why?

#

# Because the \*\*larger\*\* the absolute value of the coefficient, the more \*\*power\*\* it has to change the predicted response, resulting in a higher variance.

# ## Part 3: Regularization of linear models

#

# - Regularization is a method for "constraining" or "regularizing" the \*\*size of the coefficients\*\*, thus "shrinking" them towards zero.

# - It reduces model variance and thus \*\*minimizes overfitting\*\*.

# - If the model is too complex, it tends to reduce variance more than it increases bias, resulting in a model that is \*\*more likely to generalize\*\*.

#

# Our goal is to locate the \*\*optimum model complexity\*\*, and thus regularization is useful when we believe our model is too complex.

# ![Bias-variance tradeoff](images/bias\_variance.png)

# ### How does regularization work?

#

# For a normal linear regression model, we estimate the coefficients using the least squares criterion, which \*\*minimizes the residual sum of squares (RSS):\*\*

# ![Estimating coefficients](images/estimating\_coefficients.png)

# For a regularized linear regression model, we \*\*minimize the sum of RSS and a "penalty term"\*\* that penalizes coefficient size.

#

# \*\*Ridge regression\*\* (or "L2 regularization") minimizes: $$\text{RSS} + \alpha \sum\_{j=1}^p \beta\_j^2$$

#

# \*\*Lasso regression\*\* (or "L1 regularization") minimizes: $$\text{RSS} + \alpha \sum\_{j=1}^p |\beta\_j|$$

#

# - $p$ is the \*\*number of features\*\*

# - $\beta\_j$ is a \*\*model coefficient\*\*

# - $\alpha$ is a \*\*tuning parameter:\*\*

# - A tiny $\alpha$ imposes no penalty on the coefficient size, and is equivalent to a normal linear regression model.

# - Increasing the $\alpha$ penalizes the coefficients and thus shrinks them.

# ### Lasso and ridge path diagrams

#

# A larger alpha (towards the left of each diagram) results in more regularization:

#

# - \*\*Lasso regression\*\* shrinks coefficients all the way to zero, thus removing them from the model

# - \*\*Ridge regression\*\* shrinks coefficients toward zero, but they rarely reach zero

#

# Source code for the diagrams: [Lasso regression](http://scikit-learn.org/stable/auto\_examples/linear\_model/plot\_lasso\_lars.html) and [Ridge regression](http://scikit-learn.org/stable/auto\_examples/linear\_model/plot\_ridge\_path.html)

# ![Lasso and Ridge Path Diagrams](images/lasso\_ridge\_path.png)

# ### Advice for applying regularization

#

# \*\*Should features be standardized?\*\*

#

# - Yes, because otherwise, features would be penalized simply because of their scale.

# - Also, standardizing avoids penalizing the intercept, which wouldn't make intuitive sense.

#

# \*\*How should you choose between Lasso regression and Ridge regression?\*\*

#

# - Lasso regression is preferred if we believe many features are irrelevant or if we prefer a sparse model.

# - If model performance is your primary concern, it is best to try both.

# - ElasticNet regression is a combination of lasso regression and ridge Regression.

# ### Visualizing regularization

#

# Below is a visualization of what happens when you apply regularization. The general idea is that you are \*\*restricting the allowed values of your coefficients\*\* to a certain "region". \*\*Within that region\*\*, you want to find the coefficients that result in the best model.

# ![Lasso and Ridge Coefficient Plots](images/lasso\_ridge\_coefficients.png)

# In this diagram:

#

# - We are fitting a linear regression model with \*\*two features\*\*, $x\_1$ and $x\_2$.

# - $\hat\beta$ represents the set of two coefficients, $\beta\_1$ and $\beta\_2$, which minimize the RSS for the \*\*unregularized model\*\*.

# - Regularization restricts the allowed positions of $\hat\beta$ to the \*\*blue constraint region:\*\*

# - For lasso, this region is a \*\*diamond\*\* because it constrains the absolute value of the coefficients.

# - For ridge, this region is a \*\*circle\*\* because it constrains the square of the coefficients.

# - The \*\*size of the blue region\*\* is determined by $\alpha$, with a smaller $\alpha$ resulting in a larger region:

# - When $\alpha$ is zero, the blue region is infinitely large, and thus the coefficient sizes are not constrained.

# - When $\alpha$ increases, the blue region gets smaller and smaller.

#

# In this case, $\hat\beta$ is \*\*not\*\* within the blue constraint region. Thus, we need to \*\*move $\hat\beta$ until it intersects the blue region\*\*, while \*\*increasing the RSS as little as possible.\*\*

#

# From page 222 of [An Introduction to Statistical Learning](http://www-bcf.usc.edu/~gareth/ISL/):

#

# > The ellipses that are centered around $\hat\beta$ represent \*\*regions of constant RSS\*\*. In other words, all of the points on a given ellipse share a common value of the RSS. As the ellipses expand away from the least squares coefficient estimates, the RSS increases. Equations (6.8) and (6.9) indicate that the lasso and ridge regression coefficient estimates are given by the \*\*first point at which an ellipse contacts the constraint region\*\*.

#

# > Since \*\*ridge regression\*\* has a circular constraint with no sharp points, this intersection will not generally occur on an axis, and so the ridge regression coefficient estimates will be exclusively non-zero. However, the \*\*lasso\*\* constraint has corners at each of the axes, and so the ellipse will often intersect the constraint region at an axis. When this occurs, one of the coefficients will equal zero. In higher dimensions, many of the coefficient estimates may equal zero simultaneously. In Figure 6.7, the intersection occurs at $\beta\_1 = 0$, and so the resulting model will only include $\beta\_2$.

# ## Part 4: Regularized regression in scikit-learn

# - Communities and Crime dataset from the UCI Machine Learning Repository: [data](http://archive.ics.uci.edu/ml/machine-learning-databases/communities/communities.data), [data dictionary](http://archive.ics.uci.edu/ml/datasets/Communities+and+Crime)

# - \*\*Goal:\*\* Predict the violent crime rate for a community given socioeconomic and law enforcement data

# ### Load and prepare the crime dataset

# read in the dataset

import pandas as pd

url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/communities/communities.data'

crime = pd.read\_csv(url, header=None, na\_values=['?'])

crime.head()

# examine the response variable

crime[127].describe()

# remove categorical features

crime.drop([0, 1, 2, 3, 4], axis=1, inplace=True)

# remove rows with any missing values

crime.dropna(inplace=True)

# check the shape

crime.shape

# define X and y

X = crime.drop(127, axis=1)

y = crime[127]

# split into training and testing sets

from sklearn.cross\_validation import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=1)

# ### Linear regression

# build a linear regression model

from sklearn.linear\_model import LinearRegression

linreg = LinearRegression()

linreg.fit(X\_train, y\_train)

# examine the coefficients

print linreg.coef\_

# make predictions

y\_pred = linreg.predict(X\_test)

# calculate RMSE

from sklearn import metrics

import numpy as np

print np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))

# ### Ridge regression

#

# - [Ridge](http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.Ridge.html) documentation

# - \*\*alpha:\*\* must be positive, increase for more regularization

# - \*\*normalize:\*\* scales the features (without using StandardScaler)

# alpha=0 is equivalent to linear regression

from sklearn.linear\_model import Ridge

ridgereg = Ridge(alpha=0, normalize=True)

ridgereg.fit(X\_train, y\_train)

y\_pred = ridgereg.predict(X\_test)

print np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))

# try alpha=0.1

ridgereg = Ridge(alpha=0.1, normalize=True)

ridgereg.fit(X\_train, y\_train)

y\_pred = ridgereg.predict(X\_test)

print np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))

# examine the coefficients

print ridgereg.coef\_

# - [RidgeCV](http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.RidgeCV.html): ridge regression with built-in cross-validation of the alpha parameter

# - \*\*alphas:\*\* array of alpha values to try

# create an array of alpha values

alpha\_range = 10.\*\*np.arange(-2, 3)

alpha\_range

# select the best alpha with RidgeCV

from sklearn.linear\_model import RidgeCV

ridgeregcv = RidgeCV(alphas=alpha\_range, normalize=True, scoring='mean\_squared\_error')

ridgeregcv.fit(X\_train, y\_train)

ridgeregcv.alpha\_

# predict method uses the best alpha value

y\_pred = ridgeregcv.predict(X\_test)

print np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))

# ### Lasso regression

#

# - [Lasso](http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.Lasso.html) documentation

# - \*\*alpha:\*\* must be positive, increase for more regularization

# - \*\*normalize:\*\* scales the features (without using StandardScaler)

# try alpha=0.001 and examine coefficients

from sklearn.linear\_model import Lasso

lassoreg = Lasso(alpha=0.001, normalize=True)

lassoreg.fit(X\_train, y\_train)

print lassoreg.coef\_

# try alpha=0.01 and examine coefficients

lassoreg = Lasso(alpha=0.01, normalize=True)

lassoreg.fit(X\_train, y\_train)

print lassoreg.coef\_

# calculate RMSE (for alpha=0.01)

y\_pred = lassoreg.predict(X\_test)

print np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))

# - [LassoCV](http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LassoCV.html): lasso regression with built-in cross-validation of the alpha parameter

# - \*\*n\_alphas:\*\* number of alpha values (automatically chosen) to try

# select the best alpha with LassoCV

from sklearn.linear\_model import LassoCV

lassoregcv = LassoCV(n\_alphas=100, normalize=True, random\_state=1)

lassoregcv.fit(X\_train, y\_train)

lassoregcv.alpha\_

# examine the coefficients

print lassoregcv.coef\_

# predict method uses the best alpha value

y\_pred = lassoregcv.predict(X\_test)

print np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))

# ## Part 5: Regularized classification in scikit-learn

#

# - Wine dataset from the UCI Machine Learning Repository: [data](http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data), [data dictionary](http://archive.ics.uci.edu/ml/datasets/Wine)

# - \*\*Goal:\*\* Predict the origin of wine using chemical analysis

# ### Load and prepare the wine dataset

# read in the dataset

url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data'

wine = pd.read\_csv(url, header=None)

wine.head()

# examine the response variable

wine[0].value\_counts()

# define X and y

X = wine.drop(0, axis=1)

y = wine[0]

# split into training and testing sets

from sklearn.cross\_validation import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=1)

# ### Logistic regression (unregularized)

# build a logistic regression model

from sklearn.linear\_model import LogisticRegression

logreg = LogisticRegression(C=1e9)

logreg.fit(X\_train, y\_train)

# examine the coefficients

print logreg.coef\_

# generate predicted probabilities

y\_pred\_prob = logreg.predict\_proba(X\_test)

print y\_pred\_prob

# calculate log loss

print metrics.log\_loss(y\_test, y\_pred\_prob)

# ### Logistic regression (regularized)

#

# - [LogisticRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LogisticRegression.html) documentation

# - \*\*C:\*\* must be positive, decrease for more regularization

# - \*\*penalty:\*\* l1 (lasso) or l2 (ridge)

# standardize X\_train and X\_test

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X\_train)

X\_train\_scaled = scaler.transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# try C=0.1 with L1 penalty

logreg = LogisticRegression(C=0.1, penalty='l1')

logreg.fit(X\_train\_scaled, y\_train)

print logreg.coef\_

# generate predicted probabilities and calculate log loss

y\_pred\_prob = logreg.predict\_proba(X\_test\_scaled)

print metrics.log\_loss(y\_test, y\_pred\_prob)

# try C=0.1 with L2 penalty

logreg = LogisticRegression(C=0.1, penalty='l2')

logreg.fit(X\_train\_scaled, y\_train)

print logreg.coef\_

# generate predicted probabilities and calculate log loss

y\_pred\_prob = logreg.predict\_proba(X\_test\_scaled)

print metrics.log\_loss(y\_test, y\_pred\_prob)

# - [Pipeline](http://scikit-learn.org/stable/modules/pipeline.html): chain steps together

# - [GridSearchCV](http://scikit-learn.org/stable/modules/grid\_search.html): search a grid of parameters

# pipeline of StandardScaler and LogisticRegression

from sklearn.pipeline import make\_pipeline

pipe = make\_pipeline(StandardScaler(), LogisticRegression())

# grid search for best combination of C and penalty

from sklearn.grid\_search import GridSearchCV

C\_range = 10.\*\*np.arange(-2, 3)

penalty\_options = ['l1', 'l2']

param\_grid = dict(logisticregression\_\_C=C\_range, logisticregression\_\_penalty=penalty\_options)

grid = GridSearchCV(pipe, param\_grid, cv=10, scoring='log\_loss')

grid.fit(X, y)

# print all log loss scores

grid.grid\_scores\_

# examine the best model

print grid.best\_score\_

print grid.best\_params\_

# ## Part 6: Comparing regularized linear models with unregularized linear models

#

# \*\*Advantages of regularized linear models:\*\*

#

# - Better performance

# - L1 regularization performs automatic feature selection

# - Useful for high-dimensional problems (p > n)

#

# \*\*Disadvantages of regularized linear models:\*\*

#

# - Tuning is required

# - Feature scaling is recommended

# - Less interpretable (due to feature scaling)