Lab 1: Part 2: Understanding regularized linear regression

Devika Subramanian, ML Bootcamp, (c) 2019.

In this part, you will play with regularized linear regression (L1 and L2) and use it to study models with different bias-variance properties.

- first you will work with an artificial one-dimensional data set
- then you will work with the MLData diabetes dataset

Load up all the packages we need

In [1]:
```
import random
import numpy as np
import matplotlib.pyplot as plt

import sklearn
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import learning_curve
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV

from sklearn import linear_model
from sklearn.linear_model import Ridge
from sklearn.linear_model import LassoCV
from sklearn.linear_model import Lasso

from sklearn import datasets
from sklearn.datasets import load_diabetes

import warnings
warnings.filterwarnings("ignore")

# This is a bit of magic to make matplotlib figures appear inline in the noteb
ook
# rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# Some more magic so that the notebook will reload external python modules;
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipyt
hon
%load_ext autoreload
%autoreload 2
```

Regularized Linear Regression cost function and gradient

Below is an implementation of the loss function and gradient of the loss function for regularized linear regression on multiple predictors. Regularized linear regression has the following cost function:

$$J(\theta) = \frac{1}{2m}\left(\sum_{i=1}^{m}\left(y^{(i)} - h_\theta(x^{(i)}\right)^2\right) + \frac{\lambda}{2m}\left(\sum_{j=1}^{n}{\theta_j}^2\right)$$

where λ is a regularization parameter which controls the degree of regularization (thus, help preventing overfitting). The regularization term puts a penalty on the overall cost $J(\theta)$. As the magnitudes of the model parameters θ_j increase, the penalty increases as well. Note that you should not regularize the θ_0 term. Note the vectorized code.

Correspondingly, the partial derivative of the regularized linear regression cost function with respect to θ_j is defined as:

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}){x_j}^{(i)}$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}){x_j}^{(i)}\right) + \frac{\lambda}{m}\theta_j \quad \text{for } j \geq 1$$

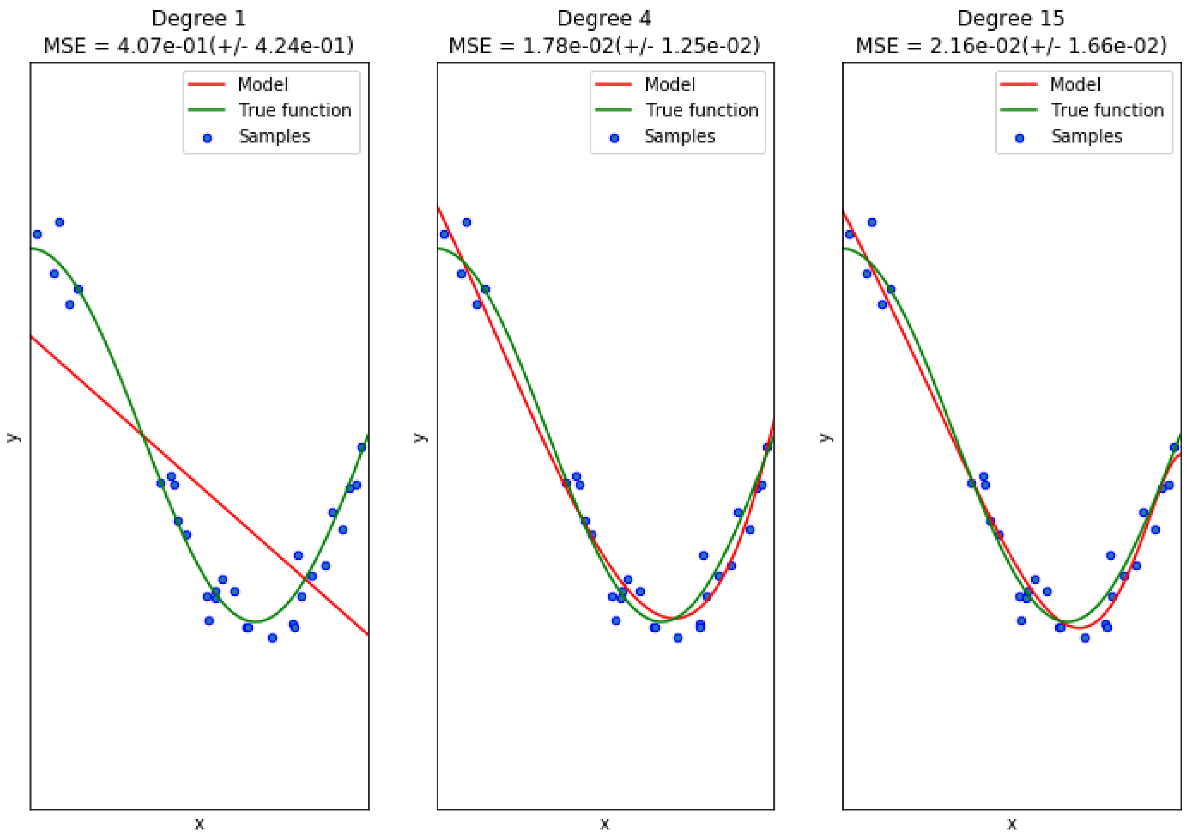
L2 Regularized Linear Regression: a simple example

In this problem, you will see the impact of regularization on model fitting. The true model is $cos(1.5\pi x)$, and the data consists of x drawn from the interval [0,1] and y is the true function with some random Gaussian noise added. The extent of regularization is determined by the parameter alpha.

Experiment with regualization strength

First set alpha = 0.0, and observe the fits. Then change alpha to 0.01, 0.1, 1.0, 10.0 and 100.0. What do you observe?

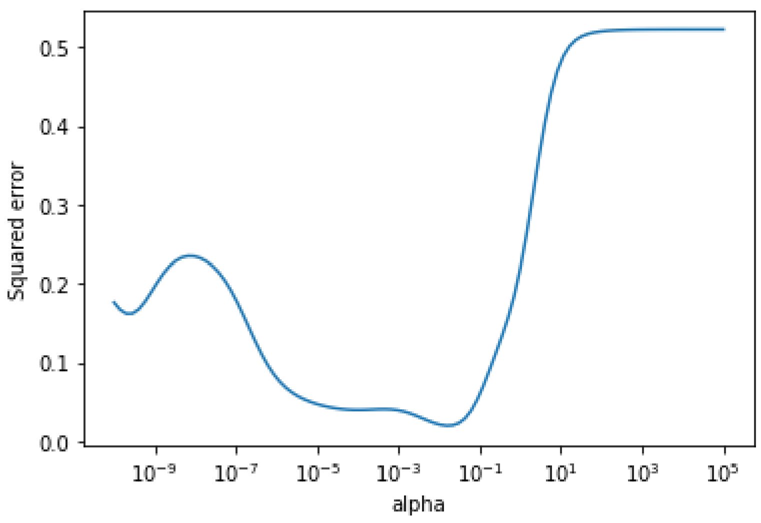
In [2]:
```
np.random.seed(0)
n_samples = 30

def true_fun(X):
    return np.cos(1.5 * np.pi * X)

X = np.sort(np.random.rand(n_samples))
y = true_fun(X) + np.random.randn(n_samples) * 0.1

degrees = [1, 4, 15]
plt.figure(figsize=(12,8))
for i in range(len(degrees)):
    ax = plt.subplot(1, len(degrees), i + 1)
    plt.setp(ax, xticks=(), yticks=())

    polynomial_features = PolynomialFeatures(degree=degrees[i],
                                             include_bias=False)
    ## CHANGE ALPHA HERE!! alpha .01 makes deg 4 look best but alpha 1.0 makes
deg 15 looks better
    ridge = Ridge(alpha=0.01)
    pipeline = Pipeline([("polynomial_features", polynomial_features),
                         ("ridge", ridge)])
    pipeline.fit(X[:, np.newaxis], y)

    # Evaluate the models using crossvalidation
    scores = cross_val_score(pipeline, X[:, np.newaxis], y,
                             scoring="neg_mean_squared_error", cv=10)

    X_test = np.linspace(0, 1, 100)
    plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Model",co
lor='r')
    plt.plot(X_test, true_fun(X_test), label="True function",color='g')
    plt.scatter(X, y, edgecolor='b', s=20, label="Samples")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.xlim((0, 1))
    plt.ylim((-2, 2))
    plt.legend(loc="best")
    plt.title("Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(
        degrees[i], -scores.mean(), scores.std()))
plt.show()
```



Find optimal alpha for L2 regression by cross validation

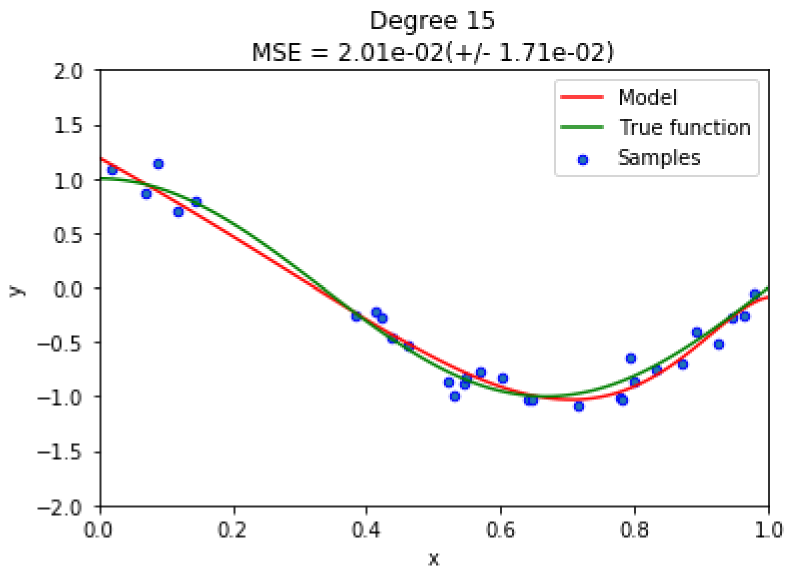
In [3]:
```
# to find the optimal value of alpha, select the highest degree you want to wo
rk with
# and then sweep through the alphas on a logarithmic scale

n_alphas = 100
alphas = np.logspace(-10, 5, n_alphas)
scores = []

for alpha in alphas:
    polynomial_features = PolynomialFeatures(degree=15,
                                             include_bias=False)
    ridge = Ridge(alpha=alpha)
    pipeline = Pipeline([("polynomial_features", polynomial_features),
                         ("ridge", ridge)])
    pipeline.fit(X[:, np.newaxis], y)
    score = cross_val_score(pipeline, X[:, np.newaxis], y,
                             scoring="neg_mean_squared_error", cv=10)
    scores.append(-np.mean(score))
    print("alpha = {0} -np.mean(score) = {1}".format(alpha, -np.mean(score)))

plt.semilogx(alphas,scores)
plt.xlabel('alpha')
plt.ylabel('Squared error')
plt.show()
```

```
alpha = 1e-10 -np.mean(score) = 0.17633928059827436
alpha = 1.4174741629268076e-10 -np.mean(score) = 0.16708406731231124
alpha = 2.0092330025650458e-10 -np.mean(score) = 0.16219462751342104
alpha = 2.848035868435805e-10 -np.mean(score) = 0.16239721174480518
alpha = 4.03701725859655e-10 -np.mean(score) = 0.1676689131791739
alpha = 5.722367659350221e-10 -np.mean(score) = 0.1770415997824466
alpha = 8.111308307896889e-10 -np.mean(score) = 0.18884294657436695
alpha = 1.1497569953977357e-09 -np.mean(score) = 0.20121284337669762
alpha = 1.6297508346206469e-09 -np.mean(score) = 0.21258971753002456
alpha = 2.310129700083158e-09 -np.mean(score) = 0.22195709338448552
alpha = 3.2745491628777316e-09 -np.mean(score) = 0.22884080626595837
alpha = 4.641588833612773e-09 -np.mean(score) = 0.2331662516409539
alpha = 6.579332246575682e-09 -np.mean(score) = 0.23508491497869125
alpha = 9.32603346883218e-09 -np.mean(score) = 0.23482644535523364
alpha = 1.3219411484660288e-08 -np.mean(score) = 0.232590215389952
alpha = 1.8738174228603867e-08 -np.mean(score) = 0.2284759037622357
alpha = 2.656087782946684e-08 -np.mean(score) = 0.2224557507534474
alpha = 3.7649358067924714e-08 -np.mean(score) = 0.21439594623369934
alpha = 5.3366992312063125e-08 -np.mean(score) = 0.20413178515227565
alpha = 7.56463327554629e-08 -np.mean(score) = 0.1915888599468461
alpha = 1.0722672220103232e-07 -np.mean(score) = 0.17692153096604799
alpha = 1.519911082952933e-07 -np.mean(score) = 0.16061458283964655
alpha = 2.1544346900318867e-07 -np.mean(score) = 0.14348263132968592
alpha = 3.0538555088334124e-07 -np.mean(score) = 0.12653317735495906
alpha = 4.3287612810830616e-07 -np.mean(score) = 0.11073445124750156
alpha = 6.135907273413176e-07 -np.mean(score) = 0.09679440983468593
alpha = 8.697490026177835e-07 -np.mean(score) = 0.08505001947809394
alpha = 1.232846739442066e-06 -np.mean(score) = 0.07549201718801185
alpha = 1.747528400007683e-06 -np.mean(score) = 0.06787705503955274
alpha = 2.477076355991714e-06 -np.mean(score) = 0.061857119121280224
alpha = 3.5111917342151347e-06 -np.mean(score) = 0.057079078475124334
alpha = 4.977023564332114e-06 -np.mean(score) = 0.053240653085149116
alpha = 7.0548023107186455e-06 -np.mean(score) = 0.05011032300397271
alpha = 1e-05 -np.mean(score) = 0.04752563635760043
alpha = 1.4174741629268048e-05 -np.mean(score) = 0.045382612724313834
alpha = 2.0092330025650458e-05 -np.mean(score) = 0.0436231172769246
alpha = 2.8480358684358048e-05 -np.mean(score) = 0.04222159064647858
alpha = 4.037017258596558e-05 -np.mean(score) = 0.04117090345848398
alpha = 5.72236765935022e-05 -np.mean(score) = 0.04046842157462883
alpha = 8.111308307896872e-05 -np.mean(score) = 0.04010366686344327
alpha = 0.00011497569953977356 -np.mean(score) = 0.04004704834912527
alpha = 0.00016297508346206434 -np.mean(score) = 0.04023791931865897
alpha = 0.00023101297000831627 -np.mean(score) = 0.04057252681827113
alpha = 0.00032745491628777317 -np.mean(score) = 0.04089731984947511
alpha = 0.0004641588833612782 -np.mean(score) = 0.0410160027357978
alpha = 0.0006579332246575682 -np.mean(score) = 0.04071561212249754
alpha = 0.0009326033468832199 -np.mean(score) = 0.03980896714209996
alpha = 0.0013219411484660286 -np.mean(score) = 0.03818335397495313
alpha = 0.0018738174228603867 -np.mean(score) = 0.03584128936799069
alpha = 0.0026560877829466894 -np.mean(score) = 0.032918277869540005
alpha = 0.0037649358067924714 -np.mean(score) = 0.029666751408416014
alpha = 0.005336699231206312 -np.mean(score) = 0.026410209602996216
alpha = 0.007564633275546291 -np.mean(score) = 0.023493951394204814
alpha = 0.010722672220103254 -np.mean(score) = 0.021270109134215398
alpha = 0.01519911082952933 -np.mean(score) = 0.020138113180762758
alpha = 0.021544346900318867 -np.mean(score) = 0.02062321081041742
alpha = 0.030538555088334123 -np.mean(score) = 0.0234345588128596
alpha = 0.043287612810830614 -np.mean(score) = 0.029416312011618023
alpha = 0.06135907273413188 -np.mean(score) = 0.039316511768826105
alpha = 0.08697490026177834 -np.mean(score) = 0.053397342345341756
alpha = 0.12328467394420685 -np.mean(score) = 0.07110025720612966
alpha = 0.1747528400007683 -np.mean(score) = 0.09111780308268116
alpha = 0.2477076355991714 -np.mean(score) = 0.11208024325915669
alpha = 0.3511191734215127 -np.mean(score) = 0.13360456819952618
alpha = 0.49770235643321137 -np.mean(score) = 0.15700094764287517
alpha = 0.705480231071866 -np.mean(score) = 0.18493120567992383
alpha = 1.0 -np.mean(score) = 0.21993096093758407
alpha = 1.4174741629268077 -np.mean(score) = 0.2625963340896979
alpha = 2.009233002565046 -np.mean(score) = 0.31065464002007953
alpha = 2.848035868435805 -np.mean(score) = 0.35959265343490243
alpha = 4.03701725859655 -np.mean(score) = 0.40442158672762557
alpha = 5.72236765935022 -np.mean(score) = 0.4414743019735405
alpha = 8.11130830789689 -np.mean(score) = 0.46931787942324654
alpha = 11.497569953977356 -np.mean(score) = 0.4885710913376773
alpha = 16.297508346206467 -np.mean(score) = 0.5010395082906195
alpha = 23.10129700083158 -np.mean(score) = 0.5087815861150753
alpha = 32.745491628777316 -np.mean(score) = 0.5135167443058577
alpha = 46.415888336127914 -np.mean(score) = 0.5164406812520592
alpha = 65.79332246575683 -np.mean(score) = 0.5182932761327483
alpha = 93.26033468832219 -np.mean(score) = 0.5195037010065058
alpha = 132.19411484660287 -np.mean(score) = 0.5203162963560649
alpha = 187.38174228603867 -np.mean(score) = 0.5208727857083304
alpha = 265.6087782946684 -np.mean(score) = 0.5212588364297399
alpha = 376.49358067924715 -np.mean(score) = 0.5215287157346815
alpha = 533.6699231206323 -np.mean(score) = 0.5217181986305588
alpha = 756.463327554629 -np.mean(score) = 0.5218515453863033
alpha = 1072.2672220103254 -np.mean(score) = 0.5219455016495245
alpha = 1519.9110829529332 -np.mean(score) = 0.5220117449071101
alpha = 2154.4346900318865 -np.mean(score) = 0.5220584641035569
alpha = 3053.8555088334124 -np.mean(score) = 0.522091418822184
alpha = 4328.7612810830615 -np.mean(score) = 0.5221146661729167
alpha = 6135.907273413189 -np.mean(score) = 0.5221310662360376
alpha = 8697.490026177835 -np.mean(score) = 0.522142636018276
alpha = 12328.467394420684 -np.mean(score) = 0.5221507982383974
alpha = 17475.28400007683 -np.mean(score) = 0.5221565565199142
alpha = 24770.76355991714 -np.mean(score) = 0.5221606188764841
alpha = 35111.91734215142 -np.mean(score) = 0.5221634847917096
alpha = 49770.23564332114 -np.mean(score) = 0.5221655066402817
alpha = 70548.0231071866 -np.mean(score) = 0.522166933015689
alpha = 100000.0 -np.mean(score) = 0.5221679392960115
```



Build final L2 regression model with best alpha

In [4]:
```
# build the final model with the optimal alpha
best_alpha = alphas[scores.index(min(scores))]

polynomial_features = PolynomialFeatures(degree=15,
                                         include_bias=False)
ridge = Ridge(alpha=best_alpha)
pipeline = Pipeline([("polynomial_features", polynomial_features),
                         ("linear_regression", ridge)])
pipeline.fit(X[:, np.newaxis], y)
score = cross_val_score(pipeline, X[:, np.newaxis], y,
                             scoring="neg_mean_squared_error", cv=10)

# plot the fit on a new test set
X_test = np.linspace(0, 1, 100)
plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Model",color=
'r')
plt.plot(X_test, true_fun(X_test), label="True function",color='g')
plt.scatter(X, y, edgecolor='b', s=20, label="Samples")
plt.xlabel("x")
plt.ylabel("y")
plt.xlim((0, 1))
plt.ylim((-2, 2))
plt.legend(loc="best")
plt.title("Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(
        15, -score.mean(), score.std()))
plt.show()
```



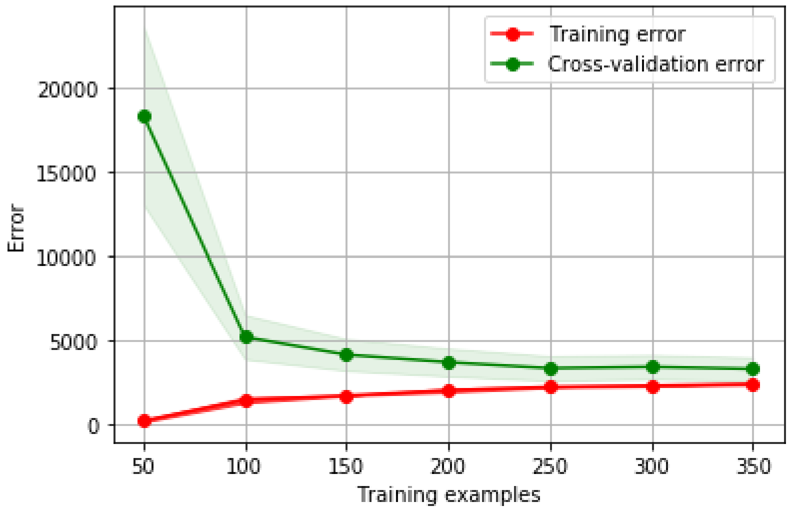
Testing overfitting and underfitting

While visualizing the best fit as shown is one possible way to debug your learning algorithm, it is not always easy to visualize the data and model, particularly if the data is high-dimensional. In the next cell, we show you how to generate learning curves that can help you debug your learning algorithm even if it is not easy to visualize the data.

The learning curve is a tool to find out how much we benefit from adding more training data and whether the estimator suffers more from a variance error or a bias error. If both the cross-validation error and the training error converge to a value that is quite high even with increasing size of the training set, we will need much more training data.

In [5]:
```
# The diabetes data set
# The full diabetes data set has 442 examples with 10 features and a real valu
ed target.

diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target
features = diabetes.feature_names

polynomial_features = PolynomialFeatures(degree=2, include_bias=True)
ridge = Ridge(alpha=0.01,normalize=True)
pipeline = Pipeline([("polynomial_features", polynomial_features),
                     ("ridge", ridge)])

tsizes = range(50,400,50)
train_sizes, train_scores, test_scores = learning_curve(pipeline,X , y, train_
sizes=tsizes,
                                                        scoring='neg_mean_squa
red_error',cv=10)
train_scores_mean = -np.mean(train_scores, axis=1)
train_scores_std = -np.std(train_scores, axis=1)
test_scores_mean = -np.mean(test_scores, axis=1)
test_scores_std = -np.std(test_scores, axis=1)

plt.xlabel("Training examples")
plt.ylabel("Error")
plt.grid(b=True)

plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.7,color="r"
)
plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training error")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation error")
plt.legend(loc="best")
```

```
Out[5]: <matplotlib.legend.Legend at 0x23823397518>
```



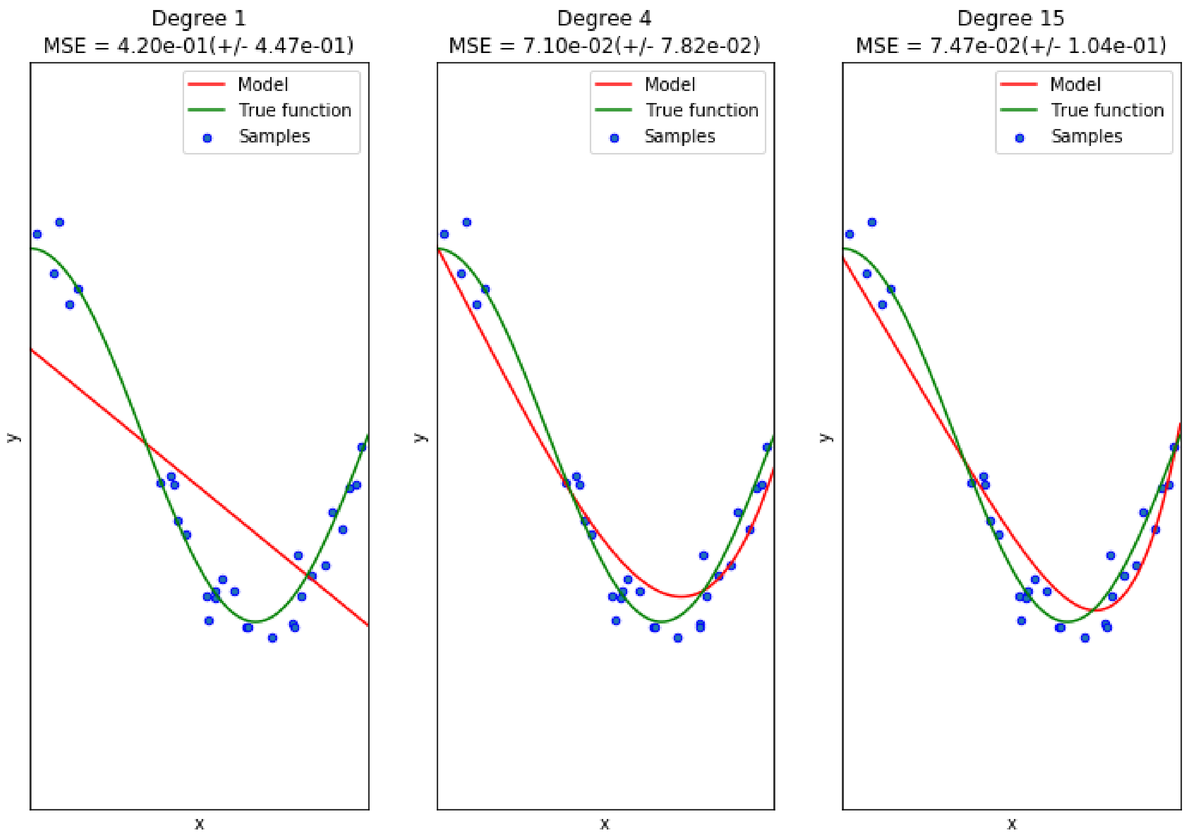
L1 regularized regression: a simple example

The true model is cos(1.5πx) , and the data consists of x drawn from the interval [0,1] and y is the true function with some random Gaussian noise added. The extent of regularization is determined by the parameter alpha.

Experiment with regualization strength

First set alpha = 0.0, and observe the fits. Then change alpha to 0.01, 0.1, 1.0, 10.0 and 100.0. What do you observe?

In [6]:
```
def true_fun(X):
    return np.cos(1.5 * np.pi * X)

np.random.seed(0)

n_samples = 30
degrees = [1, 4, 15]

X = np.sort(np.random.rand(n_samples))
y = true_fun(X) + np.random.randn(n_samples) * 0.1

plt.figure(figsize=(12,8))
for i in range(len(degrees)):
    ax = plt.subplot(1, len(degrees), i + 1)
    plt.setp(ax, xticks=(), yticks=())

    polynomial_features = PolynomialFeatures(degree=degrees[i],
                                             include_bias=False)
    ## CHANGE ALPHA HERE!!
    lasso = Lasso(alpha=0.01)
    pipeline = Pipeline([("polynomial_features", polynomial_features),
                         ("lasso", lasso)])
    pipeline.fit(X[:, np.newaxis], y)

    # Evaluate the models using crossvalidation
    scores = cross_val_score(pipeline, X[:, np.newaxis], y,
                             scoring="neg_mean_squared_error", cv=10)

    X_test = np.linspace(0, 1, 100)
    plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Model",co
lor='r')
    plt.plot(X_test, true_fun(X_test), label="True function",color='g')
    plt.scatter(X, y, edgecolor='b', s=20, label="Samples")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.xlim((0, 1))
    plt.ylim((-2, 2))
    plt.legend(loc="best")
    plt.title("Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(
        degrees[i], -scores.mean(), scores.std()))
plt.show()
```

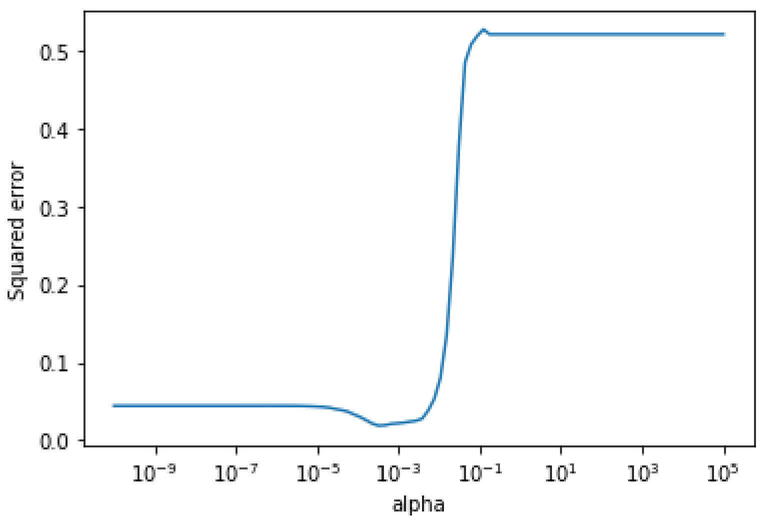


Find optimal value of alpha for L1 regularized model by cross validation

In [7]:
```
# to find the optimal value of alpha, select the highest degree you want to wo
rk with
# and then sweep through the alphas on a logarithmic scale

n_alphas = 100
alphas = np.logspace(-10, 5, n_alphas)
scores = []

for alpha in alphas:
    polynomial_features = PolynomialFeatures(degree=15,
                                             include_bias=False)
    lasso = Lasso(alpha=alpha)
    pipeline = Pipeline([("polynomial_features", polynomial_features),
                         ("lasso", lasso)])
    pipeline.fit(X[:, np.newaxis], y)
    score = cross_val_score(pipeline, X[:, np.newaxis], y,
                            scoring="neg_mean_squared_error", cv=10)
    scores.append(-np.mean(score))
    print("alpha = {0} -np.mean(score) = {1}".format(alpha, -np.mean(score)))

plt.semilogx(alphas,scores)
plt.xlabel('alpha')
plt.ylabel('Squared error')
```

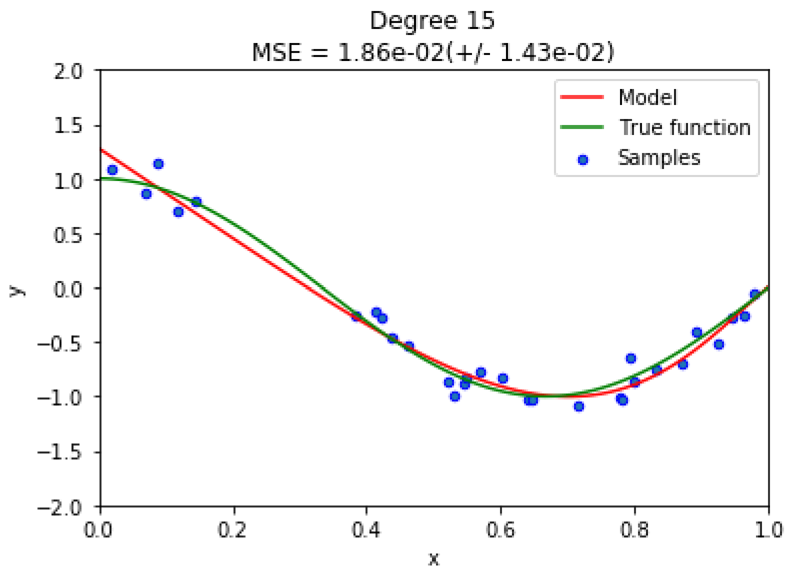
```
alpha = 1e-10 -np.mean(score) = 0.044161534285983864
alpha = 1.4174741629268076e-10 -np.mean(score) = 0.044161535498017004
alpha = 2.0092330025650458e-10 -np.mean(score) = 0.044161537216042115
alpha = 2.848035868435805e-10 -np.mean(score) = 0.04416153965129842
alpha = 4.03701725859655e-10 -np.mean(score) = 0.04416154310321366
alpha = 5.722367659350221e-10 -np.mean(score) = 0.04416154799621651
alpha = 8.111308307896889e-10 -np.mean(score) = 0.04416155493192625
alpha = 1.1497569953977357e-09 -np.mean(score) = 0.04416156476312967
alpha = 1.6297508346206469e-09 -np.mean(score) = 0.04416157869862304
alpha = 2.310129700083158e-09 -np.mean(score) = 0.04416159845187133
alpha = 3.2745491628777316e-09 -np.mean(score) = 0.0441616264516775
alpha = 4.641588833612773e-09 -np.mean(score) = 0.04416166614085457
alpha = 6.579332246575682e-09 -np.mean(score) = 0.044161722580379946
alpha = 9.32603346883218e-09 -np.mean(score) = 0.04416180240595784
alpha = 1.3219411484660288e-08 -np.mean(score) = 0.044161915554945155
alpha = 1.8738174228603867e-08 -np.mean(score) = 0.044162076466982794
alpha = 2.656087782946684e-08 -np.mean(score) = 0.04416230476448469
alpha = 3.7649358067924714e-08 -np.mean(score) = 0.04416262870843991
alpha = 5.3366992312063125e-08 -np.mean(score) = 0.04416308977106438
alpha = 7.56463327554629e-08 -np.mean(score) = 0.044163746774217194
alpha = 1.0722672220103232e-07 -np.mean(score) = 0.04416468416983381
alpha = 1.519911082952933e-07 -np.mean(score) = 0.04416602219206603
alpha = 2.1544346900318867e-07 -np.mean(score) = 0.044167931361034266
alpha = 3.0538555088334124e-07 -np.mean(score) = 0.04417067240420138
alpha = 4.3287612810830616e-07 -np.mean(score) = 0.04417461486614891
alpha = 6.135907273413176e-07 -np.mean(score) = 0.04417790741706096
alpha = 8.697490026177835e-07 -np.mean(score) = 0.04417133374541349
alpha = 1.232846739442066e-06 -np.mean(score) = 0.04416216588541353
alpha = 1.747528400007683e-06 -np.mean(score) = 0.04414925210464754
alpha = 2.477076355991714e-06 -np.mean(score) = 0.04413111732830707
alpha = 3.5111917342151347e-06 -np.mean(score) = 0.04403830920281256
alpha = 4.977023564332114e-06 -np.mean(score) = 0.043823462992558314
alpha = 7.0548023107186455e-06 -np.mean(score) = 0.043568065692577373
alpha = 1e-05 -np.mean(score) = 0.04312031594431638
alpha = 1.4174741629268048e-05 -np.mean(score) = 0.042606650297677226
alpha = 2.0092330025650458e-05 -np.mean(score) = 0.04176352726823145
alpha = 2.8480358684358048e-05 -np.mean(score) = 0.040029236510045765
alpha = 4.037017258596558e-05 -np.mean(score) = 0.03864326318993803
alpha = 5.72236765935022e-05 -np.mean(score) = 0.036677667290413656
alpha = 8.111308307896872e-05 -np.mean(score) = 0.03305786638829282
alpha = 0.00011497569953977356 -np.mean(score) = 0.029804464128808605
alpha = 0.00016297508346206434 -np.mean(score) = 0.025612291688944182
alpha = 0.00023101297000831627 -np.mean(score) = 0.021229861013482308
alpha = 0.00032745491628777317 -np.mean(score) = 0.018600444767675386
alpha = 0.0004641588833612782 -np.mean(score) = 0.01918747982386847
alpha = 0.0006579332246575682 -np.mean(score) = 0.02089862205351887
alpha = 0.0009326033468832199 -np.mean(score) = 0.02157288505373154
alpha = 0.0013219411484660286 -np.mean(score) = 0.022437991037825802
alpha = 0.0018738174228603867 -np.mean(score) = 0.023687113523803566
alpha = 0.0026560877829466894 -np.mean(score) = 0.02482524266207861
alpha = 0.0037649358067924714 -np.mean(score) = 0.02749750173703051
alpha = 0.005336699231206312 -np.mean(score) = 0.03797381924880383
alpha = 0.007564633275546291 -np.mean(score) = 0.053134190231138856
alpha = 0.010722672220103254 -np.mean(score) = 0.0803552882204824
alpha = 0.01519911082952933 -np.mean(score) = 0.1349211825709156
alpha = 0.021544346900318867 -np.mean(score) = 0.23478771685183855
alpha = 0.030538555088334123 -np.mean(score) = 0.37903107616602877
alpha = 0.043287612810830614 -np.mean(score) = 0.4860156412501924
alpha = 0.06135907273413188 -np.mean(score) = 0.5093891919609065
alpha = 0.08697490026177834 -np.mean(score) = 0.5203067651033662
alpha = 0.12328467394420685 -np.mean(score) = 0.5284763001057651
alpha = 0.1747528400007683 -np.mean(score) = 0.5221703496999568
alpha = 0.2477076355991714 -np.mean(score) = 0.5221703496999568
alpha = 0.3511191734215127 -np.mean(score) = 0.5221703496999568
alpha = 0.49770235643321137 -np.mean(score) = 0.5221703496999568
alpha = 0.705480231071866 -np.mean(score) = 0.5221703496999568
alpha = 1.0 -np.mean(score) = 0.5221703496999568
alpha = 1.4174741629268077 -np.mean(score) = 0.5221703496999568
alpha = 2.009233002565046 -np.mean(score) = 0.5221703496999568
alpha = 2.848035868435805 -np.mean(score) = 0.5221703496999568
alpha = 4.03701725859655 -np.mean(score) = 0.5221703496999568
alpha = 5.72236765935022 -np.mean(score) = 0.5221703496999568
alpha = 8.11130830789689 -np.mean(score) = 0.5221703496999568
alpha = 11.497569953977356 -np.mean(score) = 0.5221703496999568
alpha = 16.297508346206467 -np.mean(score) = 0.5221703496999568
alpha = 23.10129700083158 -np.mean(score) = 0.5221703496999568
alpha = 32.745491628777316 -np.mean(score) = 0.5221703496999568
alpha = 46.415888336127914 -np.mean(score) = 0.5221703496999568
alpha = 65.79332246575683 -np.mean(score) = 0.5221703496999568
alpha = 93.26033468832219 -np.mean(score) = 0.5221703496999568
alpha = 132.19411484660287 -np.mean(score) = 0.5221703496999568
alpha = 187.38174228603867 -np.mean(score) = 0.5221703496999568
alpha = 265.6087782946684 -np.mean(score) = 0.5221703496999568
alpha = 376.49358067924715 -np.mean(score) = 0.5221703496999568
alpha = 533.6699231206323 -np.mean(score) = 0.5221703496999568
alpha = 756.463327554629 -np.mean(score) = 0.5221703496999568
alpha = 1072.2672220103254 -np.mean(score) = 0.5221703496999568
alpha = 1519.9110829529332 -np.mean(score) = 0.5221703496999568
alpha = 2154.4346900318865 -np.mean(score) = 0.5221703496999568
alpha = 3053.8555088334124 -np.mean(score) = 0.5221703496999568
alpha = 4328.7612810830615 -np.mean(score) = 0.5221703496999568
alpha = 6135.907273413189 -np.mean(score) = 0.5221703496999568
alpha = 8697.490026177835 -np.mean(score) = 0.5221703496999568
alpha = 12328.467394420684 -np.mean(score) = 0.5221703496999568
alpha = 17475.28400007683 -np.mean(score) = 0.5221703496999568
alpha = 24770.76355991714 -np.mean(score) = 0.5221703496999568
alpha = 35111.91734215142 -np.mean(score) = 0.5221703496999568
alpha = 49770.23564332114 -np.mean(score) = 0.5221703496999568
alpha = 70548.0231071866 -np.mean(score) = 0.5221703496999568
alpha = 100000.0 -np.mean(score) = 0.5221703496999568
```

```
Out[7]: Text(0, 0.5, 'Squared error')
```



Build final L1 model with best alpha

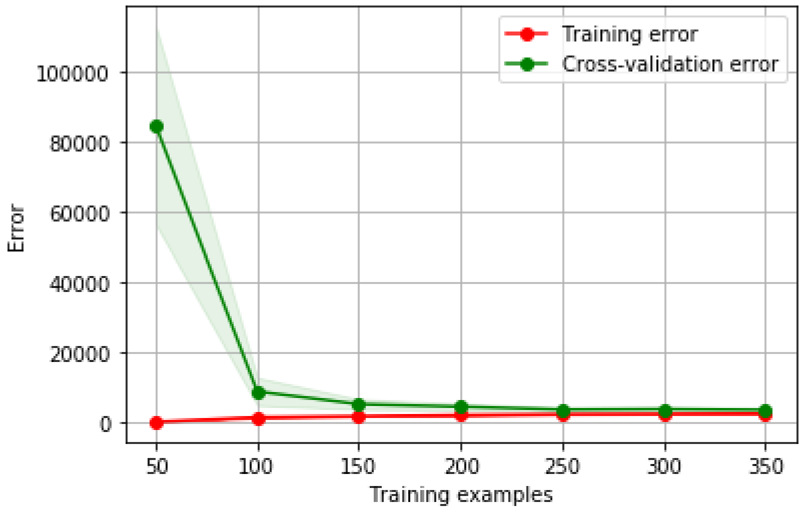
In [8]:
```
# build the final model with the optimal alpha
best_alpha = alphas[scores.index(min(scores))]

polynomial_features = PolynomialFeatures(degree=15,
                                         include_bias=False)
lasso = Lasso(alpha=best_alpha)
pipeline = Pipeline([("polynomial_features", polynomial_features),
                         ("lasso", lasso)])
pipeline.fit(X[:, np.newaxis], y)
score = cross_val_score(pipeline, X[:, np.newaxis], y,
                             scoring="neg_mean_squared_error", cv=10)

# plot the fit on a new test set
X_test = np.linspace(0, 1, 100)
plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Model",color=
'r')
plt.plot(X_test, true_fun(X_test), label="True function",color='g')
plt.scatter(X, y, edgecolor='b', s=20, label="Samples")
plt.xlabel("x")
plt.ylabel("y")
plt.xlim((0, 1))
plt.ylim((-2, 2))
plt.legend(loc="best")
plt.title("Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(
        15, -score.mean(), score.std()))
plt.show()
```



Running Lasso on the Diabetes data set

In [9]:
```
# The diabetes data set
# The full diabetes data set has 442 examples with 10 features and a real valu
ed target.

diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target
features = diabetes.feature_names

polynomial_features = PolynomialFeatures(degree=2, include_bias=True)
lasso = Lasso(alpha=0.001,normalize=True)
pipeline = Pipeline([("polynomial_features", polynomial_features),
                     ("lasso", lasso)])

tsizes = range(50,400,50)
train_sizes, train_scores, test_scores = learning_curve(pipeline,X , y, train_
sizes=tsizes,
                                                        scoring='neg_mean_squa
red_error',cv=10)
train_scores_mean = -np.mean(train_scores, axis=1)
train_scores_std = -np.std(train_scores, axis=1)
test_scores_mean = -np.mean(test_scores, axis=1)
test_scores_std = -np.std(test_scores, axis=1)

plt.xlabel("Training examples")
plt.ylabel("Error")
plt.grid(b=True)

plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.7,color="r"
)
plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training error")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation error")
plt.legend(loc="best")
```

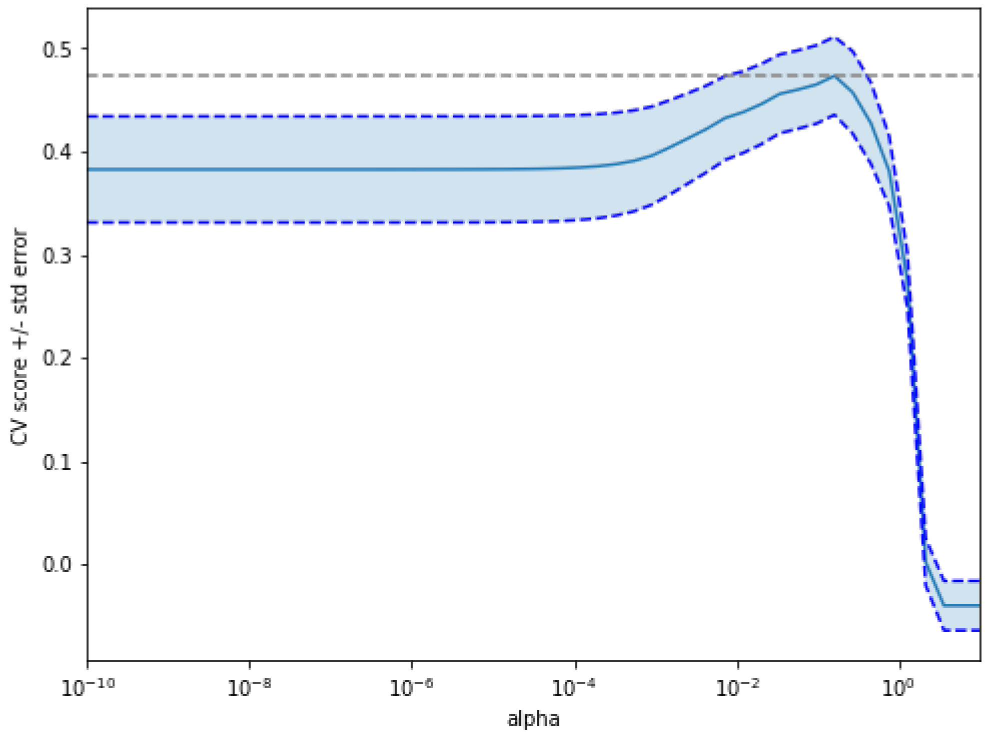
```
Out[9]: <matplotlib.legend.Legend at 0x23822140128>
```



Find best configuration for L1 model for diabetes data set

In [10]:
```
# Comparing models built by L1 and L2 regularization on Diabetes dataset

diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target

polynomial_features = PolynomialFeatures(degree=2, include_bias=True)
lasso = Lasso(normalize=True)
pipeline = Pipeline([("polynomial_features", polynomial_features),
                     ("lasso", lasso)])

alphas = np.logspace(-10, 1, 50)
tuned_parameters = [{'polynomial_features__degree':[2],
                    'lasso__alpha':alphas}]
n_folds = 10

clf = GridSearchCV(pipeline, tuned_parameters, cv=n_folds, refit=False)
clf.fit(X, y)

scores = clf.cv_results_['mean_test_score']
scores_std = clf.cv_results_['std_test_score']

# plot the results

plt.figure().set_size_inches(8, 6)
plt.semilogx(alphas, scores)

# plot error lines showing +/- std. errors of the scores
std_error = scores_std / np.sqrt(n_folds)

plt.semilogx(alphas, scores + std_error, 'b--')
plt.semilogx(alphas, scores - std_error, 'b--')

# alpha=0.2 controls the translucency of the fill color
plt.fill_between(alphas, scores + std_error, scores - std_error, alpha=0.2)
plt.ylabel('CV score +/- std error')
plt.xlabel('alpha')
plt.axhline(np.max(scores), linestyle='--', color='.5')
plt.xlim([alphas[0], alphas[-1]])

# best is max
```

```
Out[10]: (1e-10, 10.0)
```



Fit best L1 model and see coefficients

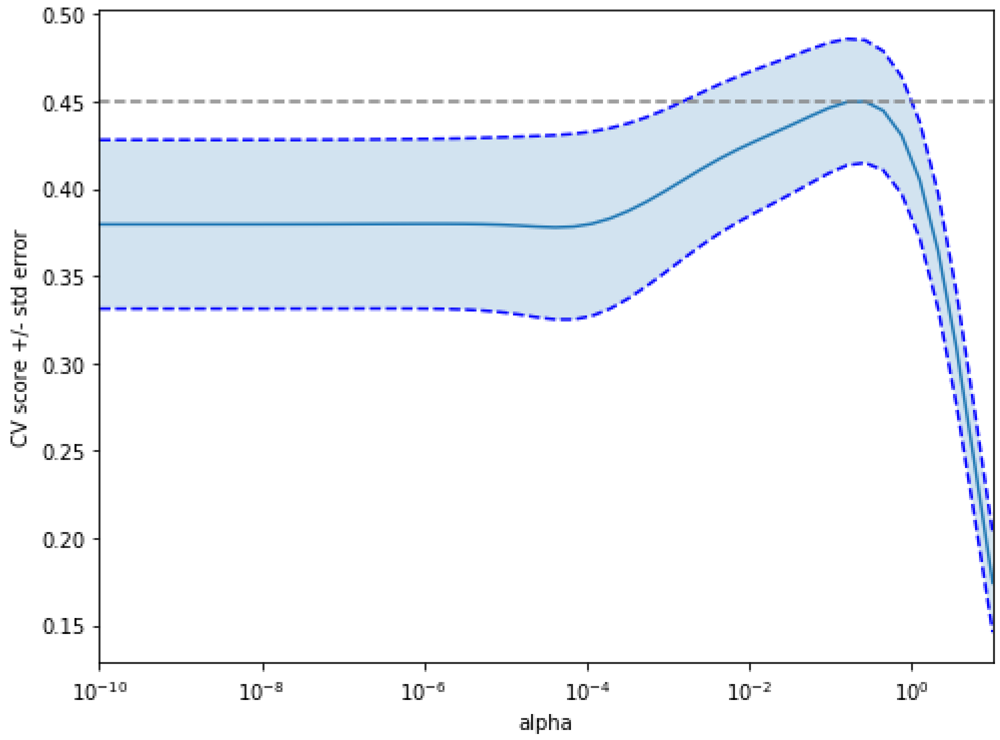
In [11]:
```
best_alpha = alphas[np.argmax(scores)]
polynomial_features = PolynomialFeatures(degree=2, include_bias=True)
lasso = Lasso(normalize=True,alpha=best_alpha)
pipeline = Pipeline([("polynomial_features", polynomial_features),
                     ("lasso", lasso)])
pipeline.fit(X,y)
score = cross_val_score(pipeline, X, y,
                             scoring="neg_mean_squared_error", cv=10)
print(-np.mean(score))
print(lasso.coef_, best_alpha)
```

```
2953.3021633143344
[ 0.00000000e+00  0.00000000e+00 -2.12798212e-01  5.02135615e+02
  2.44998077e+02 -0.00000000e+00 -0.00000000e+00 -1.79377518e+02
  0.00000000e+00  4.66188046e+02  1.32711387e+01  1.80545794e+01
  2.15994801e+03  0.00000000e+00  6.57667304e+02 -0.00000000e+00
 -0.00000000e+00  0.00000000e+00  0.00000000e+00  1.30425700e+02
  3.04893422e+02 -1.64501776e+04  0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00  0.00000000e+00 -0.00000000e+00
  0.00000000e+00  0.00000000e+00  5.19106364e+02  1.61862299e+03
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00  0.00000000e+00 -0.00000000e+00
  0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00  0.00000000e+00 -0.00000000e+00
  0.00000000e+00 -0.00000000e+00  0.00000000e+00 -0.00000000e+00
  0.00000000e+00  9.12134177e+02] 0.15998587196060574
```

Find best configuration for L2 model for diabetes data set

In [12]:
```
diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target

polynomial_features = PolynomialFeatures(degree=2, include_bias=True)
ridge = Ridge(normalize=True)
pipeline = Pipeline([("polynomial_features", polynomial_features),
                     ("ridge", ridge)])

alphas = np.logspace(-10, 1, 50)
tuned_parameters = [{'polynomial_features__degree':[2],
                    'ridge__alpha':alphas}]
n_folds = 10

clf = GridSearchCV(pipeline, tuned_parameters, cv=n_folds, refit=False)
clf.fit(X, y)

scores = clf.cv_results_['mean_test_score']
scores_std = clf.cv_results_['std_test_score']

# plot the results

plt.figure().set_size_inches(8, 6)
plt.semilogx(alphas, scores)

# plot error lines showing +/- std. errors of the scores
std_error = scores_std / np.sqrt(n_folds)

plt.semilogx(alphas, scores + std_error, 'b--')
plt.semilogx(alphas, scores - std_error, 'b--')

# alpha=0.2 controls the translucency of the fill color
plt.fill_between(alphas, scores + std_error, scores - std_error, alpha=0.2)
plt.ylabel('CV score +/- std error')
plt.xlabel('alpha')
plt.axhline(np.max(scores), linestyle='--', color='.5')
plt.xlim([alphas[0], alphas[-1]])
```

```
Out[12]: (1e-10, 10.0)
```



Fit best L2 model and see coefficients

In [13]:
```
best_alpha = alphas[np.argmax(scores)]
polynomial_features = PolynomialFeatures(degree=2, include_bias=True)
ridge = Ridge(normalize=True,alpha=best_alpha)
pipeline = Pipeline([("polynomial_features", polynomial_features),
                     ("ridge", ridge)])
pipeline.fit(X,y)
score = cross_val_score(pipeline, X, y,
                             scoring="neg_mean_squared_error", cv=10)
print(-np.mean(score))
print(ridge.coef_, best_alpha)
```

```
3101.5738389923363
[ 0.00000000e+00  5.01496236e+01 -9.22422122e+01  3.89885271e+02
  2.73143913e+02 -1.39924035e+01 -6.17965569e+01 -1.85551728e+02
  1.35949507e+02  3.77730036e+02  1.01428081e+02  1.06882451e+03
  2.45755120e+03 -3.70326713e+02  9.50020469e+02 -4.10382238e+02
 -1.50996430e+03  1.04330404e+03  1.90787693e+02  1.49204191e+03
  9.15047848e+02 -1.52757280e+04  1.01373098e+03  1.25089289e+03
  3.35887291e+02 -6.56353800e+02  1.44421219e+03 -5.66657275e+02
  1.15009002e+02  4.40868827e+02  1.25403168e+03  2.20188177e+03
 -7.53050502e+02 -4.11010869e+02  1.16566705e+02 -9.07849525e+01
  2.77296448e+02  7.06279648e+02  1.57392982e+02  5.39575145e+02
  4.56647364e+02  6.81002106e+02 -6.76324473e+02  3.96046177e+02
 -1.26690032e+03  2.26848222e+02  2.53948944e+02  5.86790893e+02
 -1.06252413e+03 -4.25689387e+02  4.71979097e+01 -2.18113728e+02
 -1.31620732e+02 -1.50640013e+02  1.40240288e+03  2.79614570e+02
 -1.42792588e+02 -6.87740064e+02  6.91569401e+02  4.52119264e+02
  5.02725995e+02 -9.47113735e+02  1.02082158e+03 -4.23508358e+02
  1.10750834e+02  1.00756954e+03] 0.26826957952797276
```

In [ ]:

In [ ]: