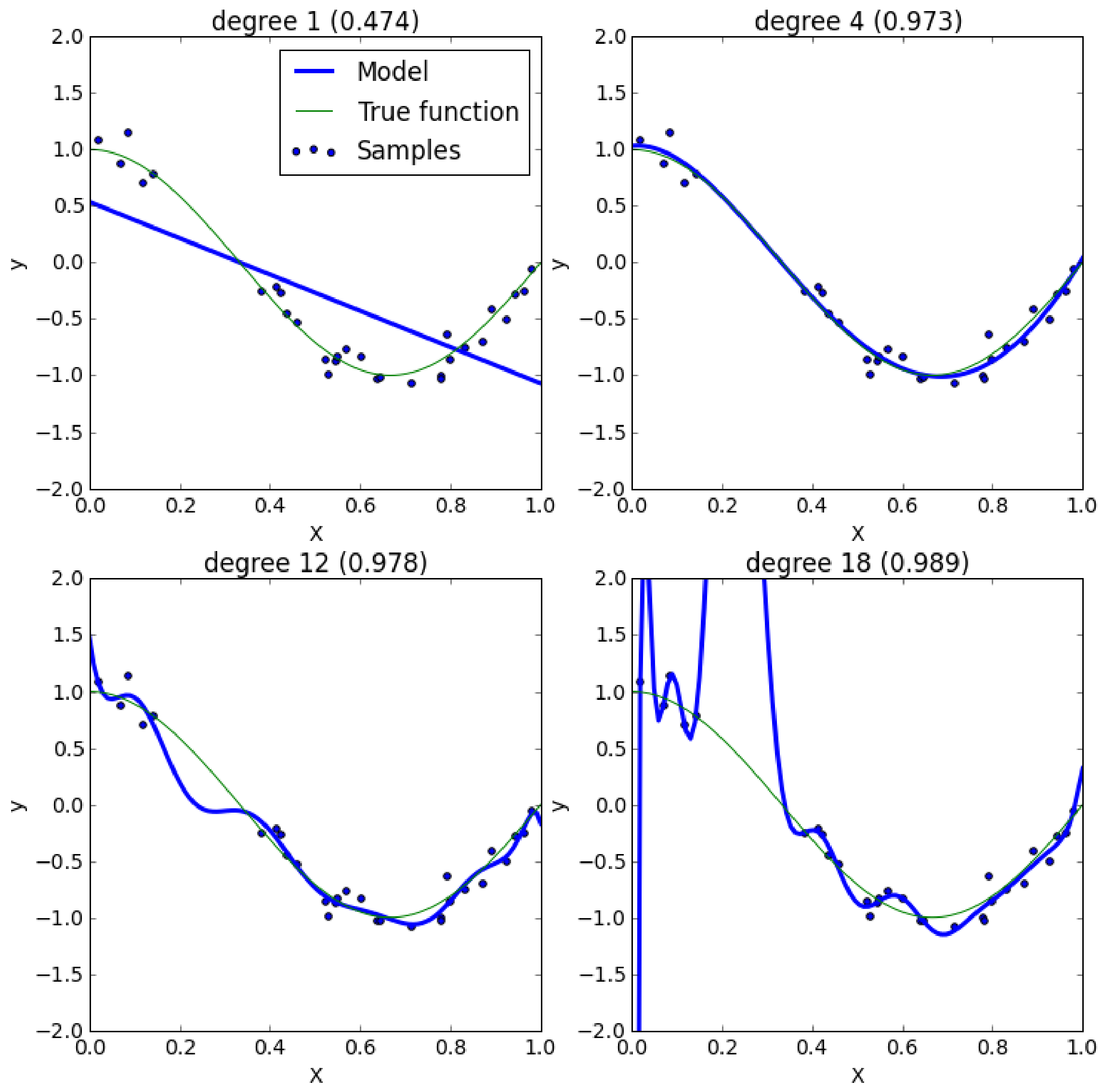
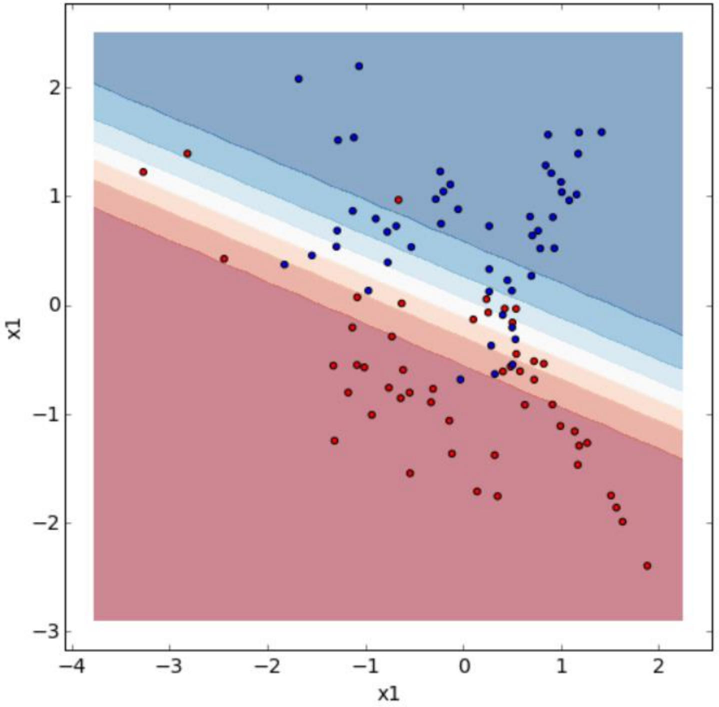
regularization

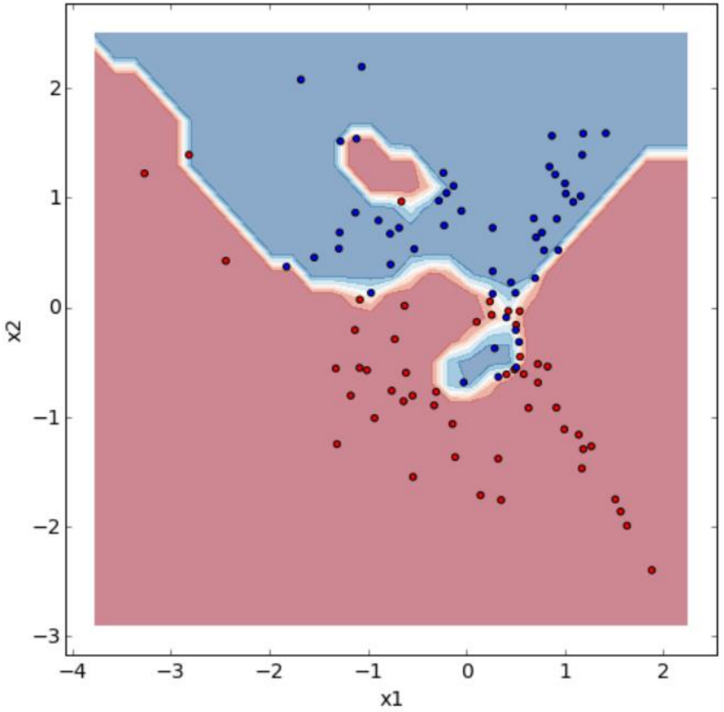


- noise
  - label
  - feature
- feature relevance
- relatively small train sets
- don’t try to fit the data perfectly
- don’t try to use all features
- notice how R^2 on the train set does increase with *d*

regularization



d = 1
accuracy = 84%



d = 7
accuracy = 97%

- noise
  - label
  - feature
- feature relevance
- relatively small train sets
- don’t try to fit the data perfectly
- don’t try to use all features
- notice how the accuracy on the train set increases with *d*

regularized linear regression

$$f(x,\theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m$$

cost function

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(f(x^{(i)},\theta) - y^{(i)})^2$$

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

regularized cost function

$$\theta_0 := \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}(f(x^{(i)},\theta) - y^{(i)})x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha\frac{1}{n}\sum_{i=1}^{n}(f(x^{(i)},\theta) - y^{(i)})x_j^{(i)} - \frac{\lambda}{n}\theta_j$$

regularized logistic regression

$$f(x,\theta) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m)$$

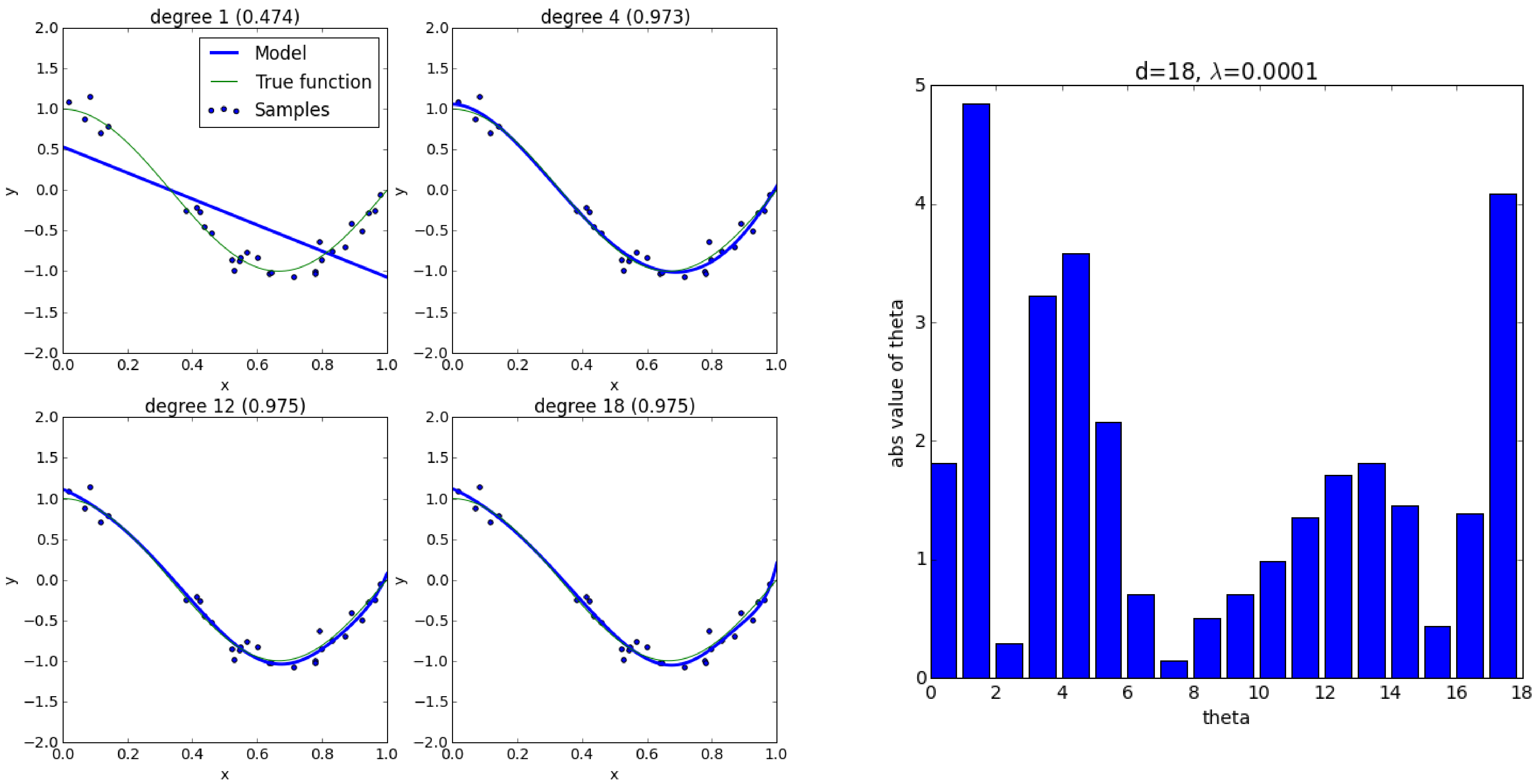
cost function

$$J(\theta) = -[\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)},\theta)) + (1-y^{(i)}) log(1-f(x^{(i)},\theta))]$$

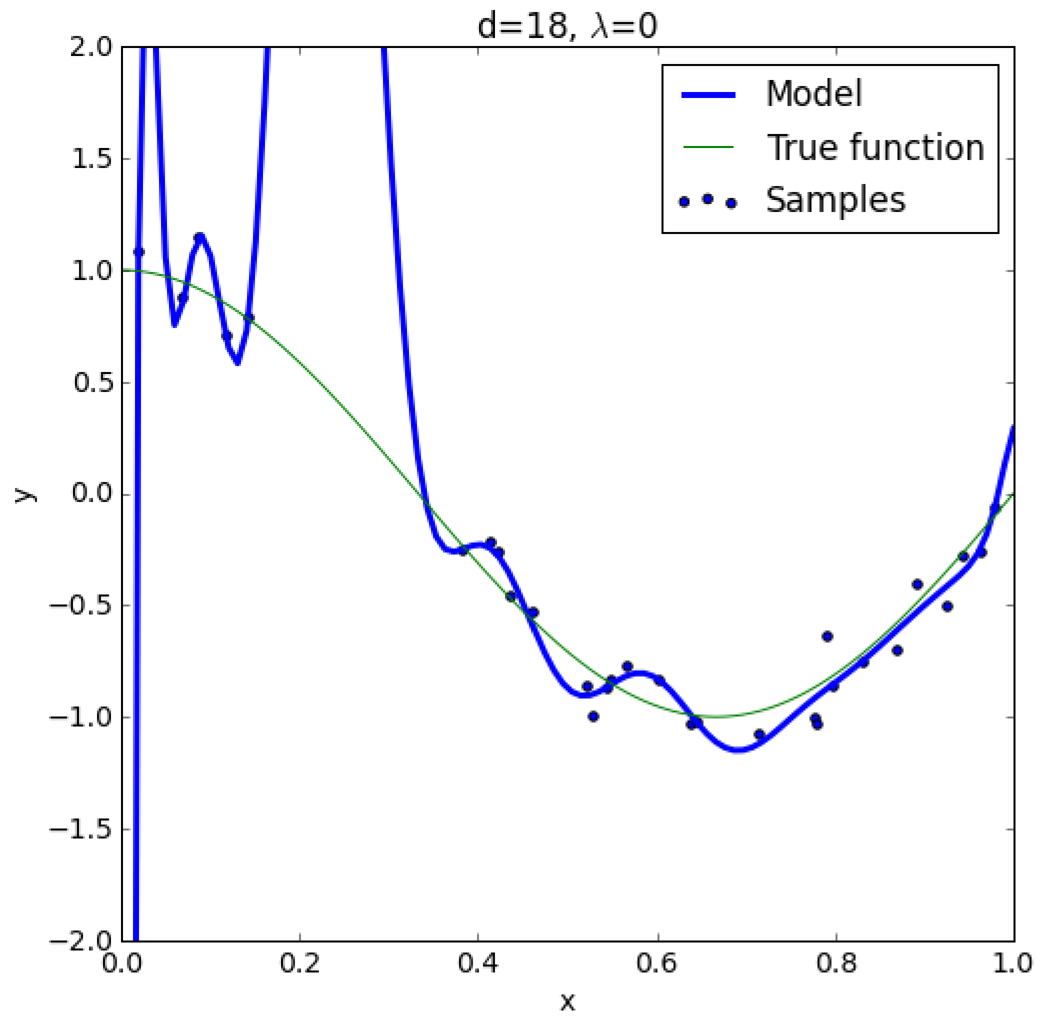
$$J(\theta) = -[\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)},\theta)) + (1-y^{(i)}) log(1-f(x^{(i)},\theta))] + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

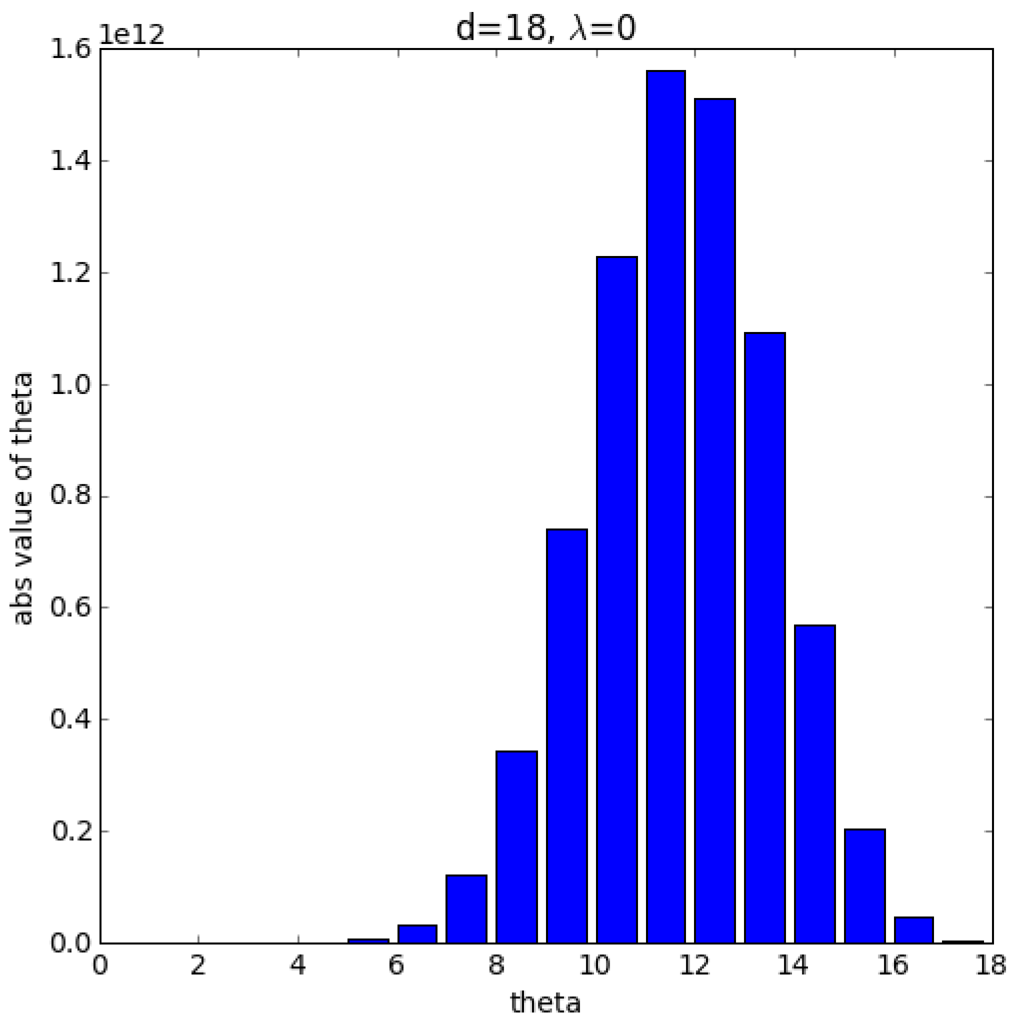
regularized cost function

regularized linear regression

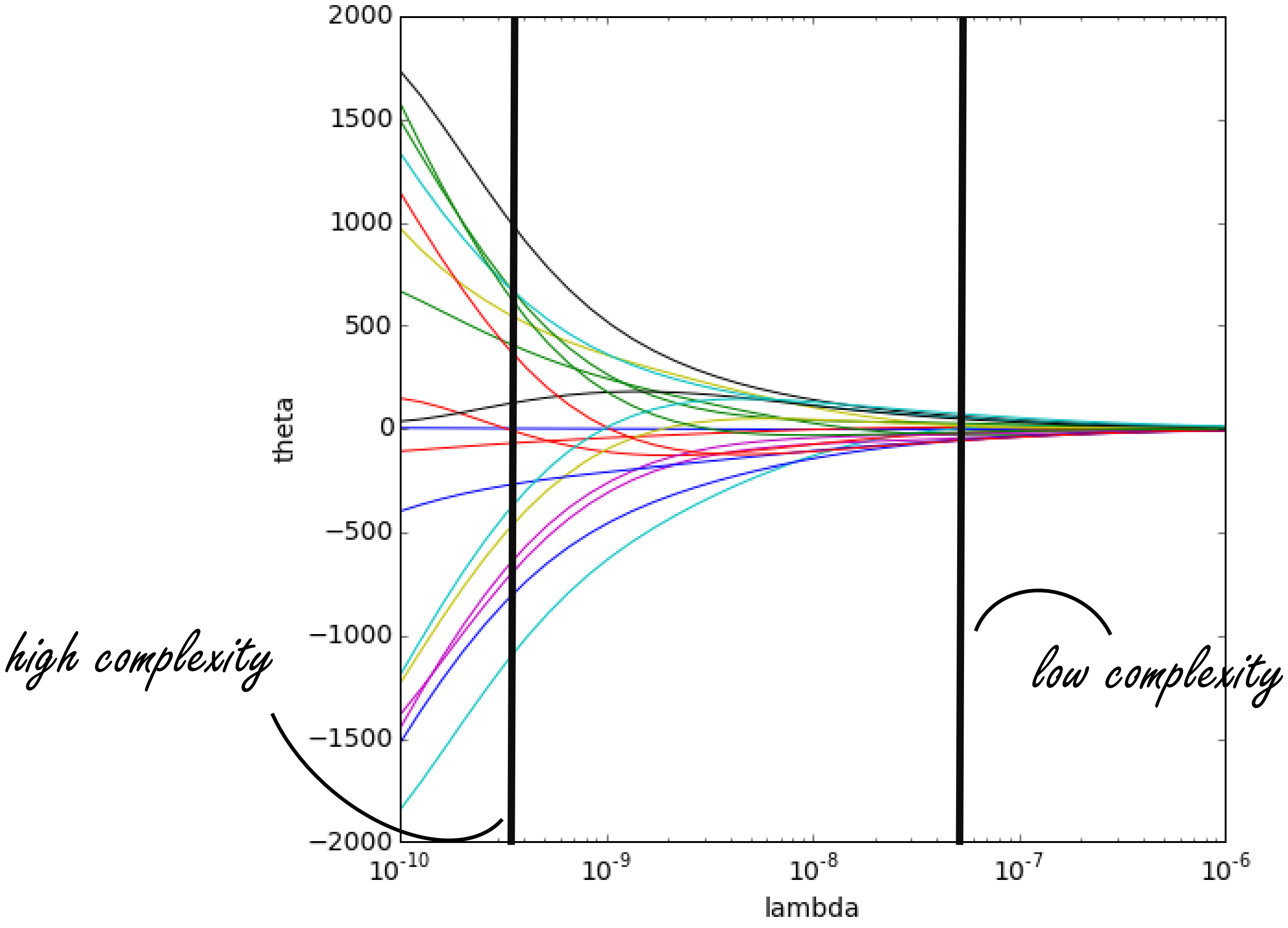


regularized linear regression

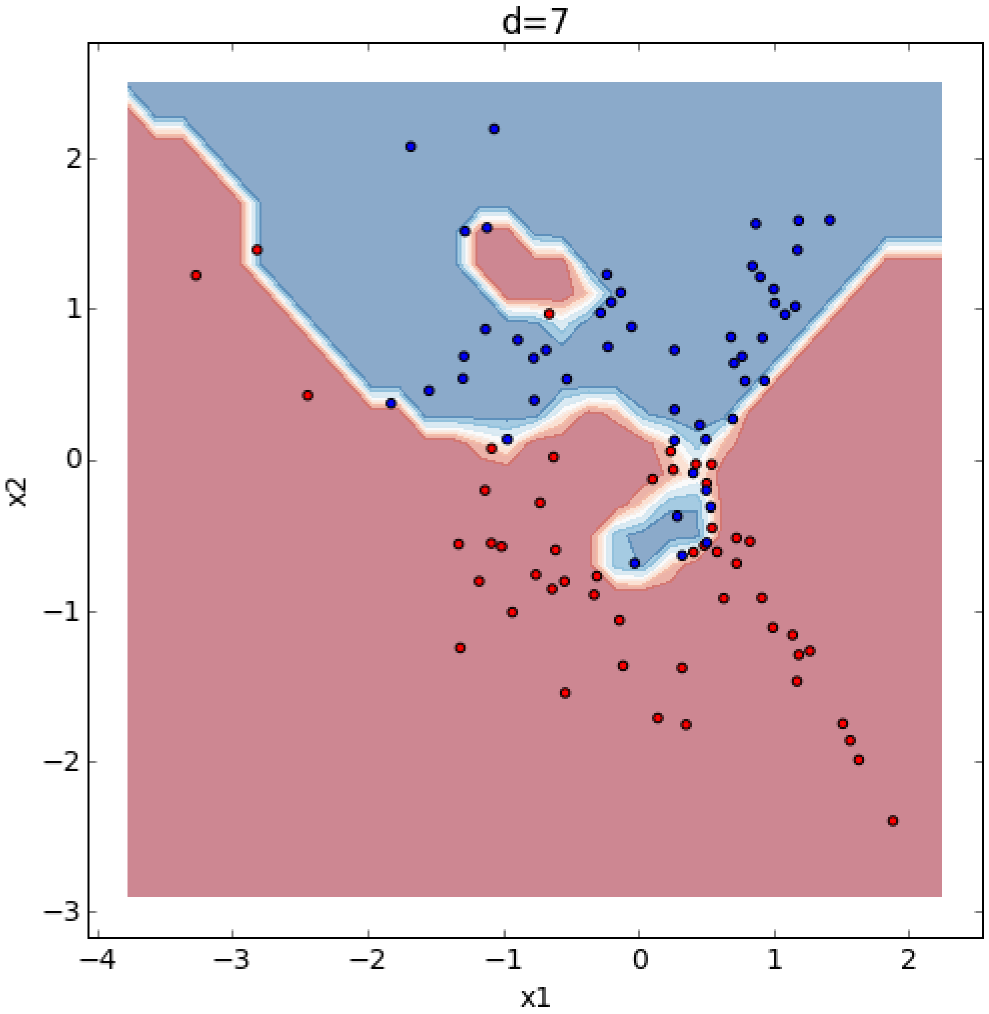


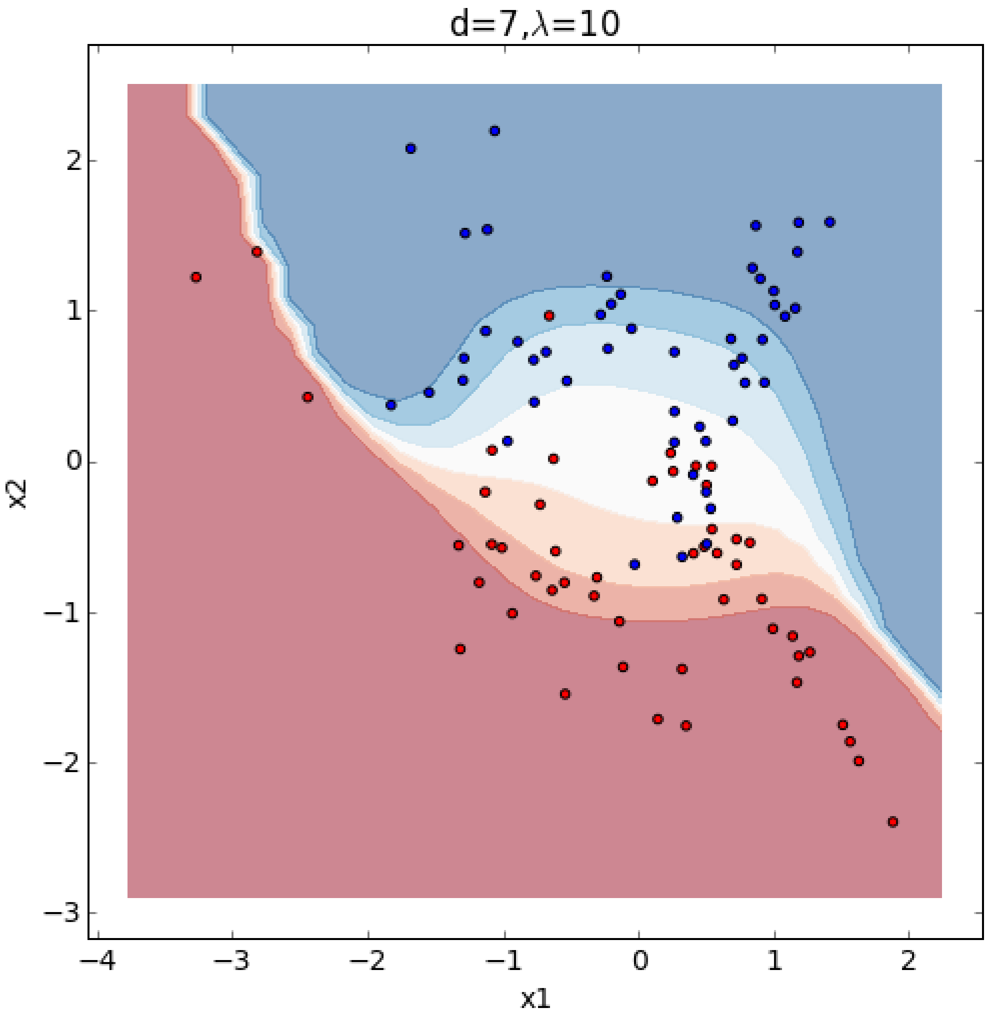


regularized linear regression

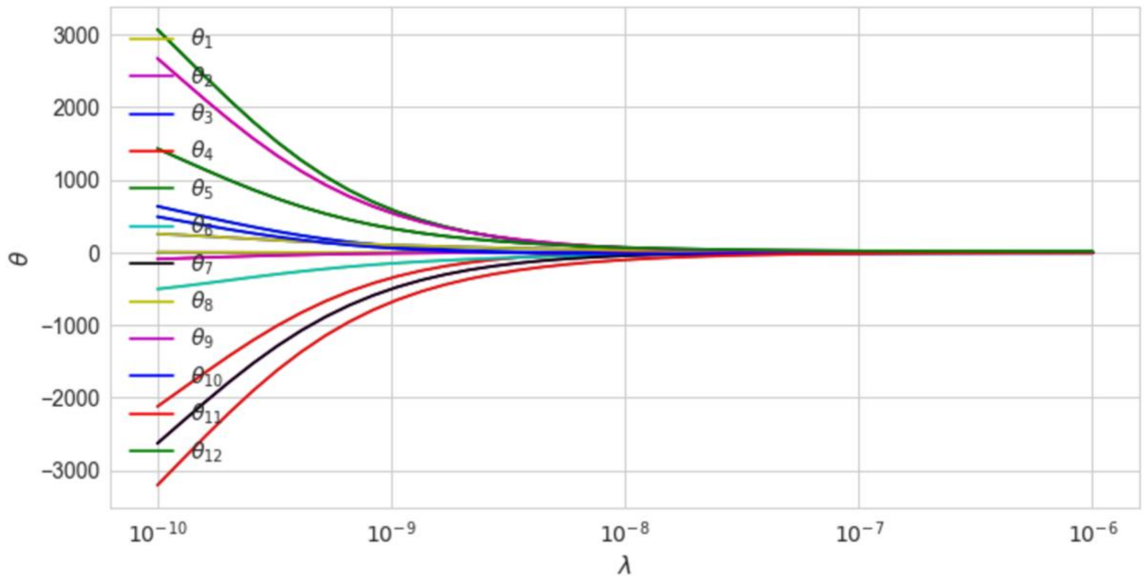


regularized logistic regression

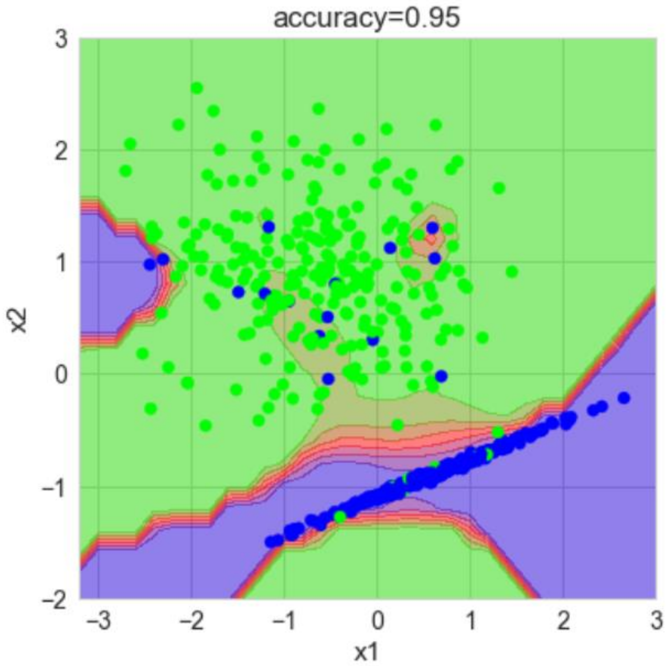


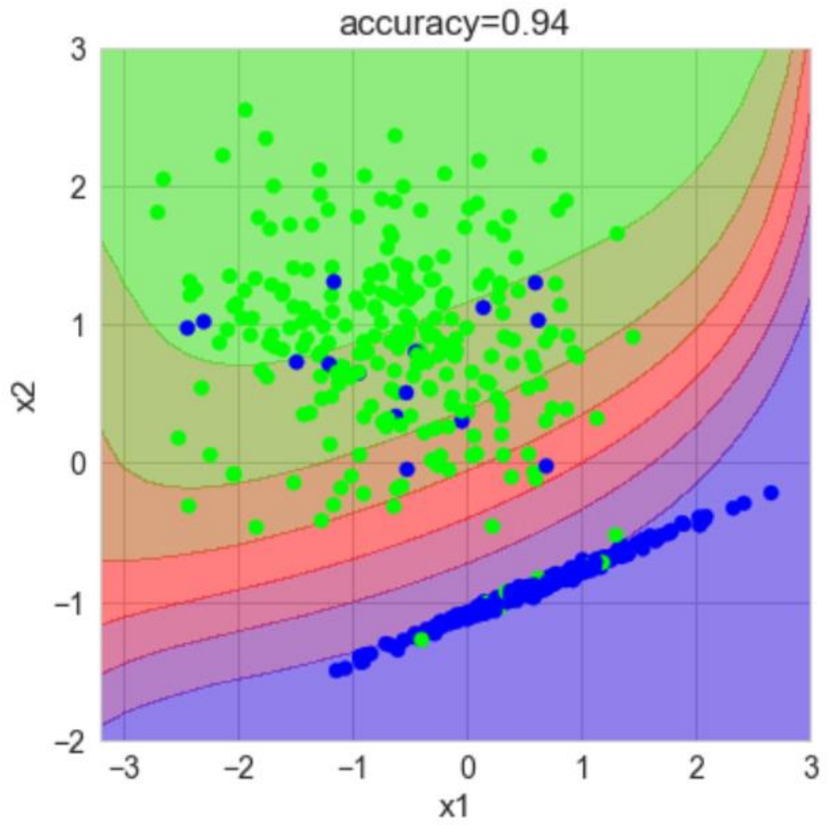


```
n_lambda = 40
lambdas = np.logspace(-6, -10, n_lambda)

model_ridge = Ridge()

coefs = []
for l in lambdas:
    model_ridge.set_params(alpha=l)
    model_ridge.fit(X, y)
    coefs.append(model_ridge.coef_)

plt.figure(figsize=(12,6))
compomics_import.plot_coefs(lambdas,coefs)
plt.show()
```



```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import MinMaxScaler

feature_scaler = MinMaxScaler()

clf = LogisticRegression(C=1000000)
polynomial_features = PolynomialFeatures(degree=7)
model_pipeline = Pipeline([("polynomial_features", polynomial_features),
                           ("MinMaxScaler", feature_scaler),
                         ("logistic_regression", clf)])

model_pipeline.fit(X,y)

plt.figure(figsize=(6,6))
compomics_import.plot_decision_boundary(model_pipeline,X,y)
plt.show()
```





```python
from sklearn.grid_search import GridSearchCV

search_space = np.logspace(-10, 10, 10, base=2)

params = dict(logistic_regression__C=search_space)
grid_search = GridSearchCV(model_pipeline, param_grid=params)

grid_search.fit(X, y)
```

```python
for params, mean_score, scores in grid_search.grid_scores_:
    print("%0.3f (+/-%0.03f) for %r" % (mean_score, scores.std() * 2, params))
```

```
0.908 (+/-0.056) for {'logistic_regression__C': 0.0009765625}
0.928 (+/-0.045) for {'logistic_regression__C': 0.0045567540608442061}
0.936 (+/-0.054) for {'logistic_regression__C': 0.021262343752724643}
0.942 (+/-0.037) for {'logistic_regression__C': 0.099212565748012488}
0.938 (+/-0.049) for {'logistic_regression__C': 0.46293735614364534}
0.936 (+/-0.046) for {'logistic_regression__C': 2.1601194777846118}
0.928 (+/-0.040) for {'logistic_regression__C': 10.079368399158989}
0.922 (+/-0.049) for {'logistic_regression__C': 47.031503752819212}
0.920 (+/-0.045) for {'logistic_regression__C': 219.45445961038678}
0.920 (+/-0.045) for {'logistic_regression__C': 1024.0}
```

```python
from sklearn import metrics
from sklearn.model_selection import cross_val_predict
from sklearn.grid_search import GridSearchCV

params = dict(logistic_regression__C=search_space)
grid_search = GridSearchCV(model_pipeline, param_grid=params)

cv_predictions = cross_val_predict(grid_search, X, y, method="predict_proba")
print cv_predictions
```

```python
fpr, tpr, thresholds = metrics.roc_curve(y, cv_predictions[:,1])
print metrics.auc(fpr, tpr)
```

```
0.993304825237
```