15-388/688 - Practical Data Science: Linear classification

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Outline

Example: classifying tumors

Classification in machine learning

Example classification algorithms

Libraries for machine learning

Outline

Example: classifying tumors

Classification in machine learning

Example classification algorithms

Libraries for machine learning

Classification tasks

Regression tasks: predicting real-valued quantity $y \in \mathbb{R}$

Classification tasks: predicting *discrete-valued* quantity y

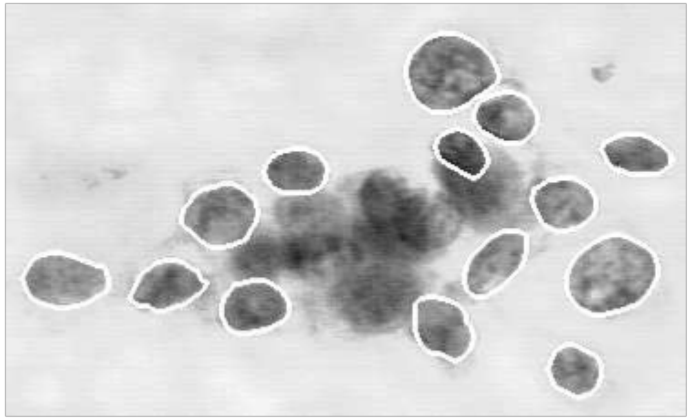
Binary classification: $y \in \{-1, +1\}$

Multiclass classification: $y \in \{1,2, \ldots, k\}$

Example: breast cancer classification

Well-known classification example: using machine learning to diagnose whether a breast tumor is benign or malignant [Street et al., 1992]

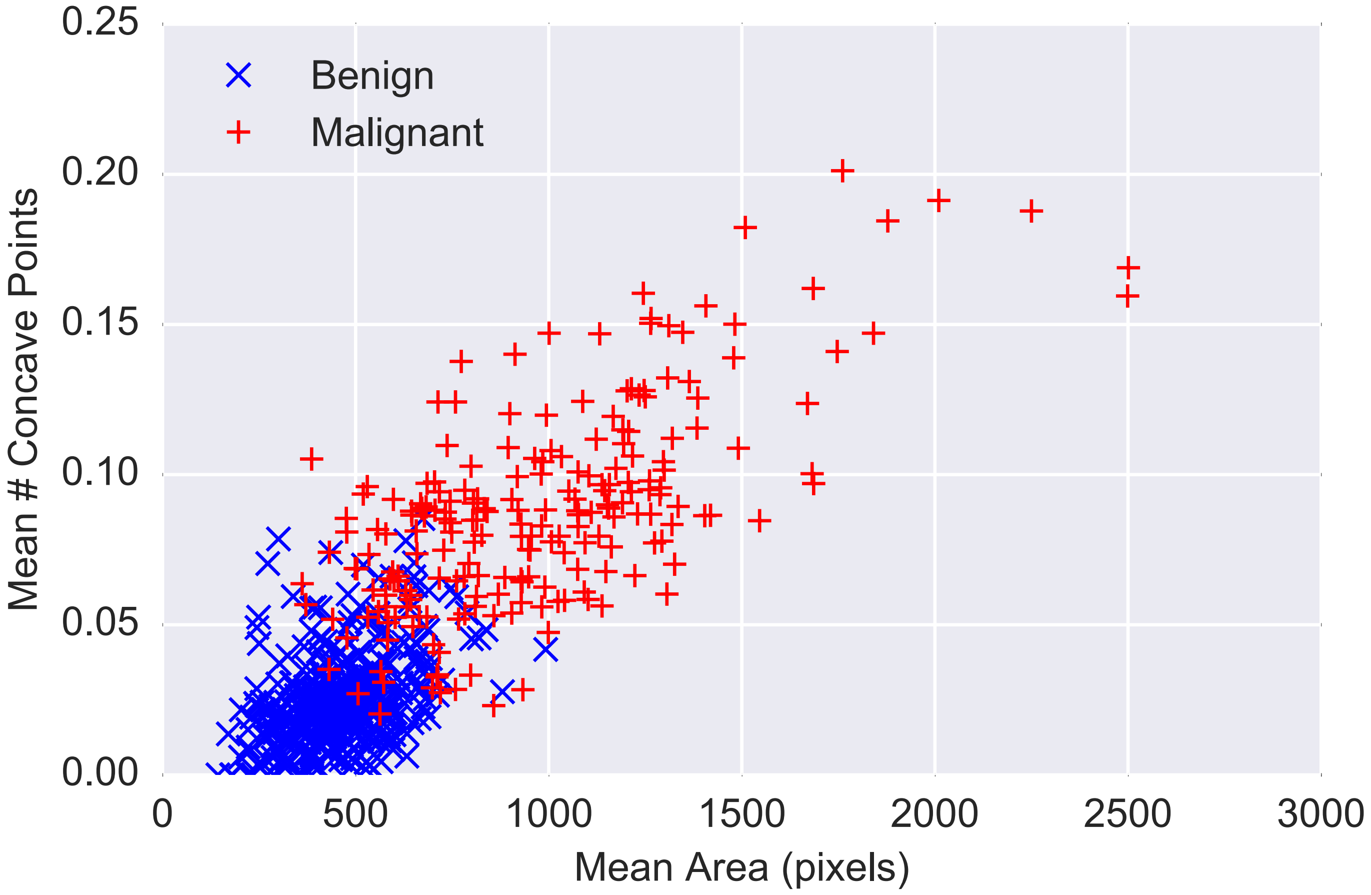
Setting: doctor extracts a sample of fluid from tumor, stains cells, then outlines several of the cells (image processing refines outline)



System computes features for each cell such as area, perimeter, concavity, texture (10 total); computes mean/std/max for all features

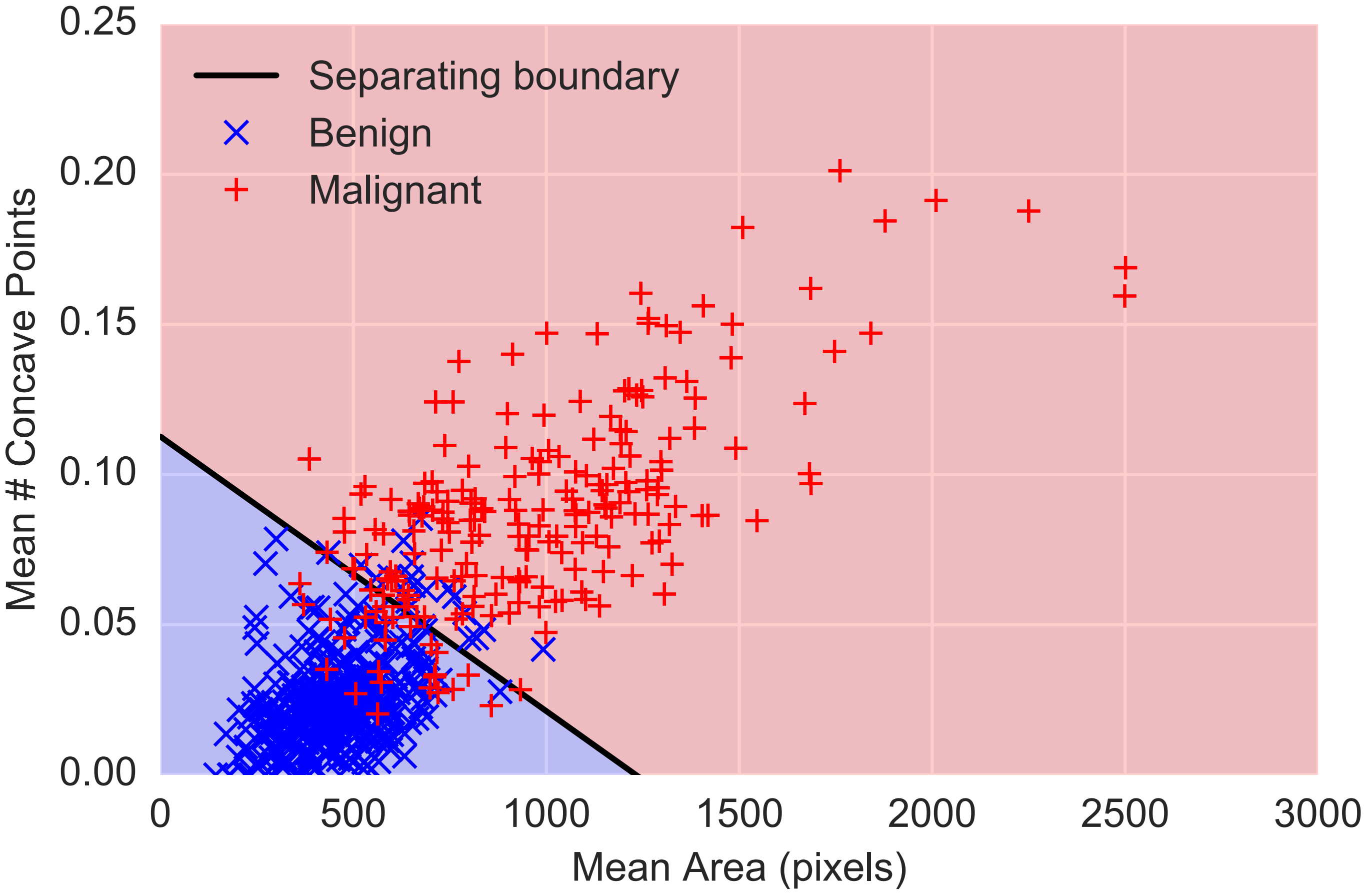
Example: breast cancer classification

Plot of two features: mean area vs. mean concave points, for two classes



Linear classification example

Linear classification $\equiv$ “drawing line separating classes”



Outline

Example: classifying tumors

Classification in machine learning

Example classification algorithms

Libraries for machine learning

Formal setting

Input features: $x^{(i)} \in \mathbb{R}^n, i = 1, \dots, m$

$$\mathrm{E.g.:}\;\; x^{(i)} = \begin{bmatrix} \mathrm{Mean\_Area}^{(i)} \\ \mathrm{Mean\_Concave\_Points}^{(i)} \\ 1 \end{bmatrix}$$

Outputs: $y^{(i)} \in \mathcal{Y},\;\; i = 1, \dots, m$

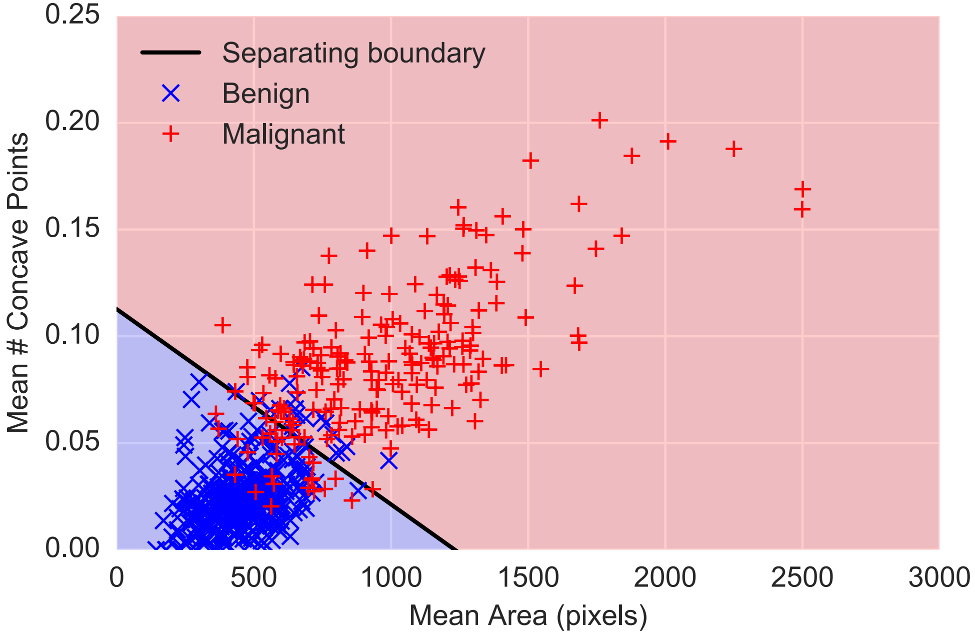
$$\mathrm{E.g.:}\;\; y^{(i)} \in \{-1 \;(\mathrm{benign}), +1 \;(\mathrm{malignant})\}$$

Model parameters: $\theta \in \mathbb{R}^n$

Hypothesis function: $h_\theta : \mathbb{R}^n \rightarrow \mathbb{R}$, aims for same *sign* as the output (informally, a measure of *confidence* in our prediction)

$$\mathrm{E.g.:}\;\; h_\theta(x) = \theta^T x, \qquad \hat{y} = \mathrm{sign}(h_\theta(x))$$

Understanding linear classification diagrams



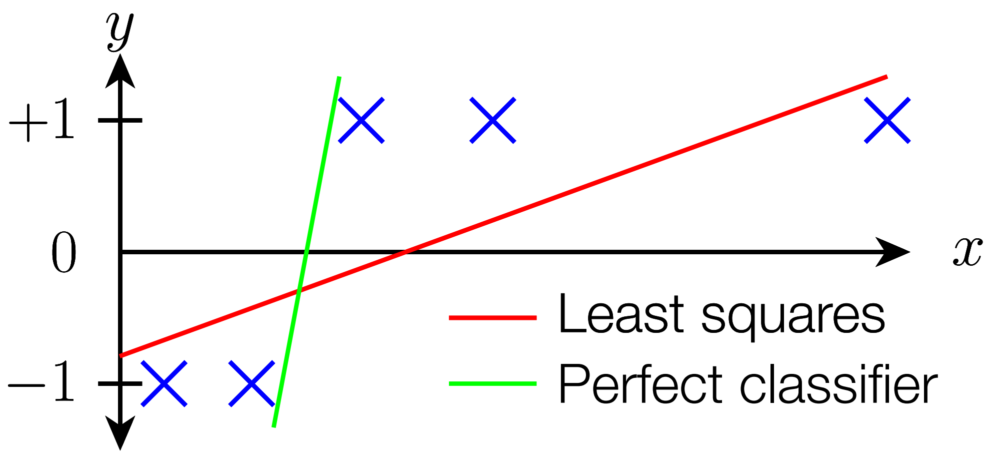
Color shows regions where the $h_\theta(x)$ is positive

Separating boundary is given by the equation $h_\theta(x) = 0$

Loss functions for classification

How do we define a loss function $\ell\colon \mathbb{R} \times \{-1, +1\} \to \mathbb{R}_+$?

What about just using squared loss?

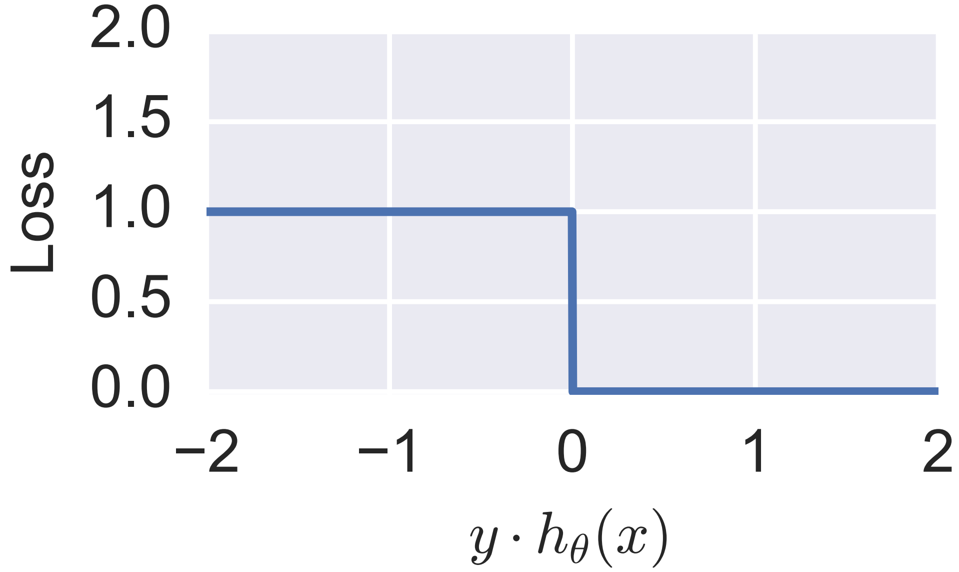


0/1 loss (i.e. error)

The loss we would like to minimize (0/1 loss, or just “error”):

$$\ell_{0/1}(h_\theta(x), y) = \begin{cases} 0 & \text{if } \operatorname{sign}\big(h_\theta(x)\big) = y \\ 1 & \text{otherwise} \end{cases}$$

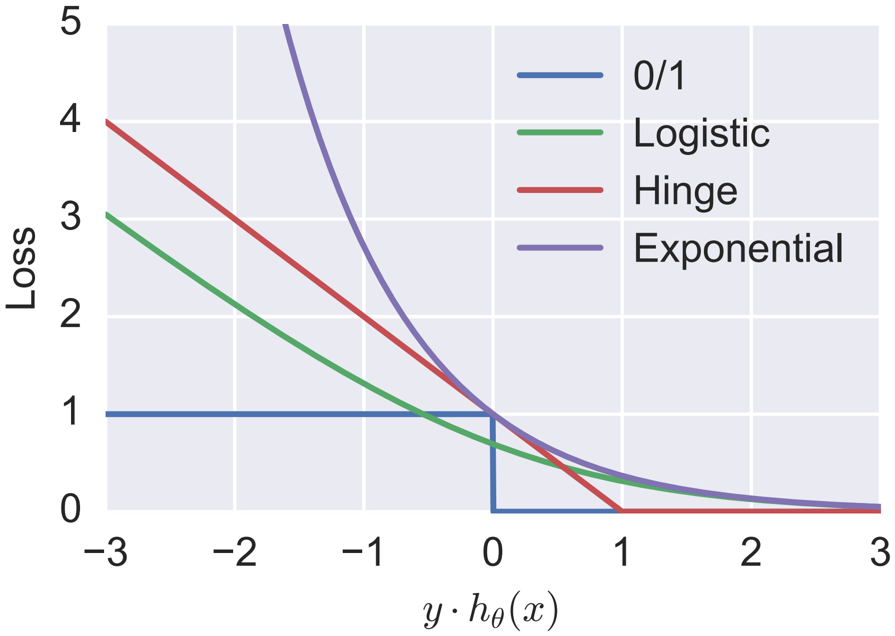
$$= \mathbf{1}\{y \cdot h_\theta(x) \leq 0\}$$



Alternative losses

Unfortunately 0/1 loss is hard to optimize (NP-hard to find classifier with minimum 0/1 loss, relates to a property called convexity of the function)

A number of alternative losses for classification are typically used instead



$$\ell_{0/1} = 1\{y \cdot h_\theta(x) \leq 0\}$$
$$\ell_{\text{logistic}} = \log(1 + \exp(-y \cdot h_\theta(x)))$$
$$\ell_{\text{hinge}} = \max\{1 - y \cdot h_\theta(x), 0\}$$
$$\ell_{\exp} = \exp(-y \cdot h_\theta(x))$$

Poll: sensitivity to outliers

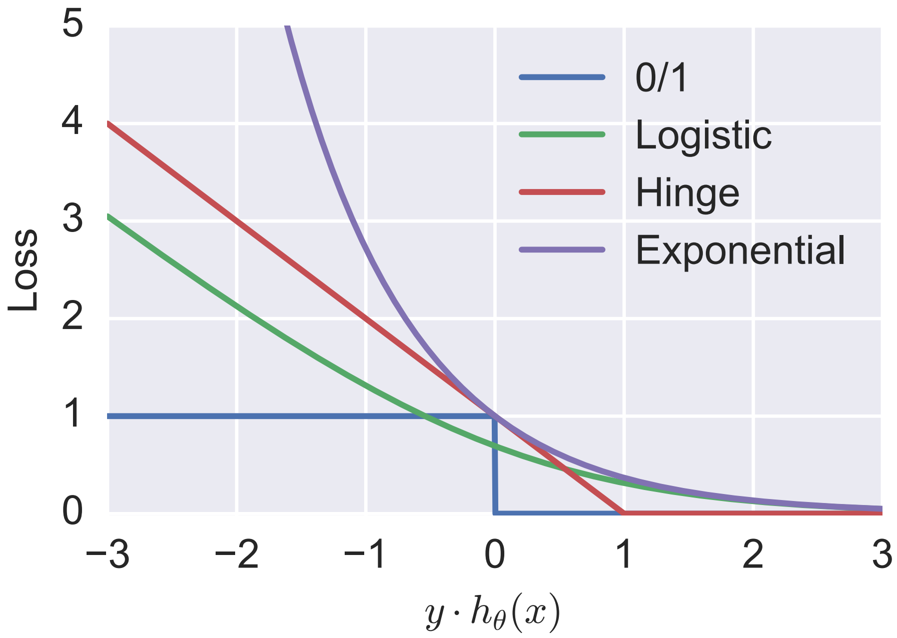
How sensitive would you estimate each of the following losses would be to outliers (i.e., points typically heavily misclassified)?

1. 0/1 < Exp < {Hinge,Logitistic}

2. Exp < Hinge < Logistic < 0/1

3. Hinge < 0/1 < Logistic < Exp

4. 0/1 < {Hinge,Logistic} < Exp



5. Outliers don't exist in the classification because output space is bounded

Machine learning optimization

With this notation, the “canonical” machine learning problem is written in the exact same way

$$\operatorname*{minimize}_{\theta} \sum_{i=1}^{m} \ell(h_\theta(x^{(i)}), y^{(i)})$$

Unlike least squares, there is not an analytical solution to the zero gradient condition for most classification losses

Instead, we solve these optimization problems using gradient descent (or a alternative optimization method, but we’ll only consider gradient descent here)

$$\text{Repeat: } \theta := \theta - \alpha \sum_{i=1}^{m} \nabla_\theta \ell(\, h_\theta(x^{(i)}), y^{(i)})$$

Outline

Example: classifying tumors

Classification in machine learning

Example classification algorithms

Libraries for machine learning

Support vector machine

A (linear) support vector machine (SVM) just solves the canonical machine learning optimization problem using hinge loss and linear hypothesis, plus an additional regularization term, more on this next lecture

$$\operatorname*{minimize}_{\theta} \sum_{i=1}^{m} \max\{1 - y^{(i)} \cdot \theta^T x^{(i)}, 0\} + \frac{\lambda}{2}\|\theta\|_2^2$$

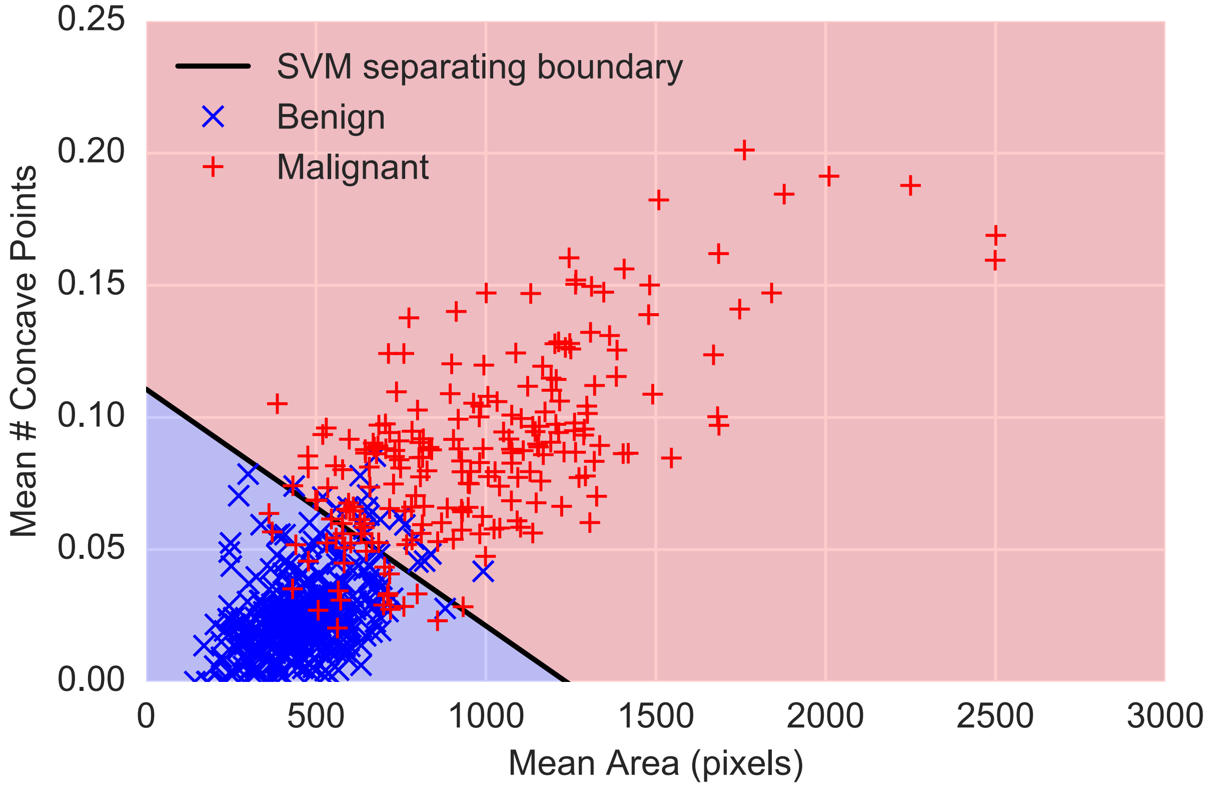
Even more precisely, the "standard" SVM doesn't actually regularize the θ_i corresponding to the constant feature, but we'll ignore this here

Updates using gradient descent:

$$\theta := \theta \ - \alpha \sum_{i=1}^{m} -y^{(i)} x^{(i)} 1\{ y^{(i)} \cdot \theta^T x^{(i)} \leq 1\} - \alpha\lambda\theta$$

Support vector machine example

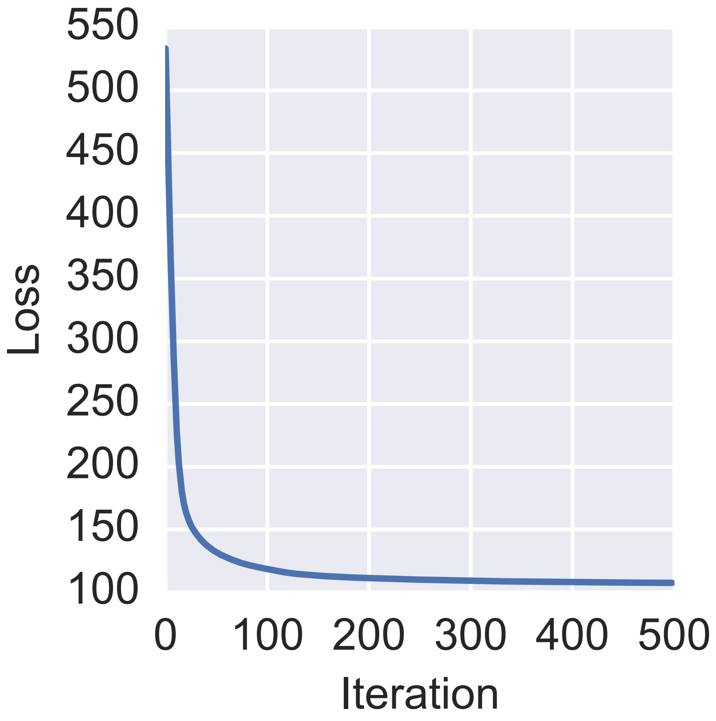
Running support vector machine on cancer dataset, with small regularization parameter (effectively zero)

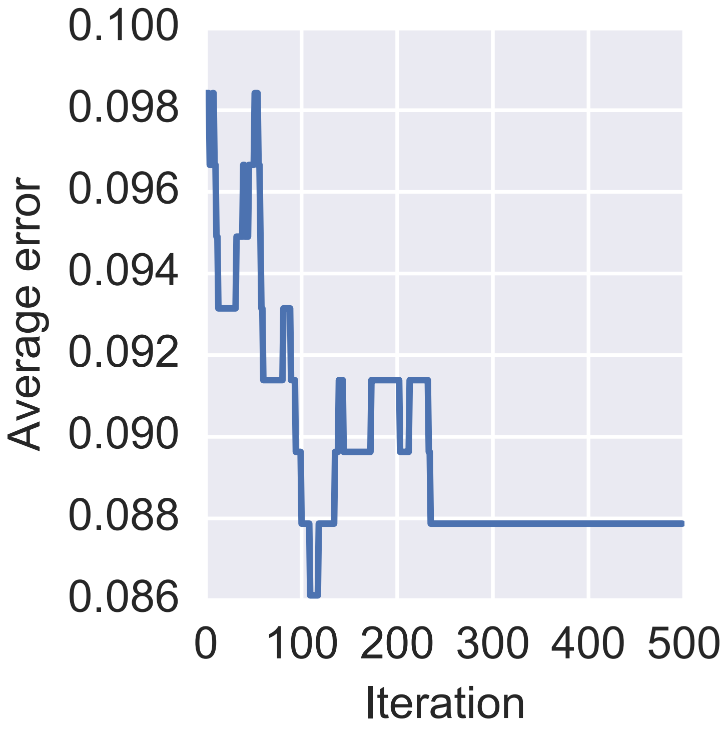


$$\theta = \begin{bmatrix} 1.456 \\ 1.848 \\ -0.189 \end{bmatrix}$$

SVM optimization progress

Optimization objective and error versus gradient descent iteration number





Logistic regression

Logistic regression just solves this problem using logistic loss and linear hypothesis function

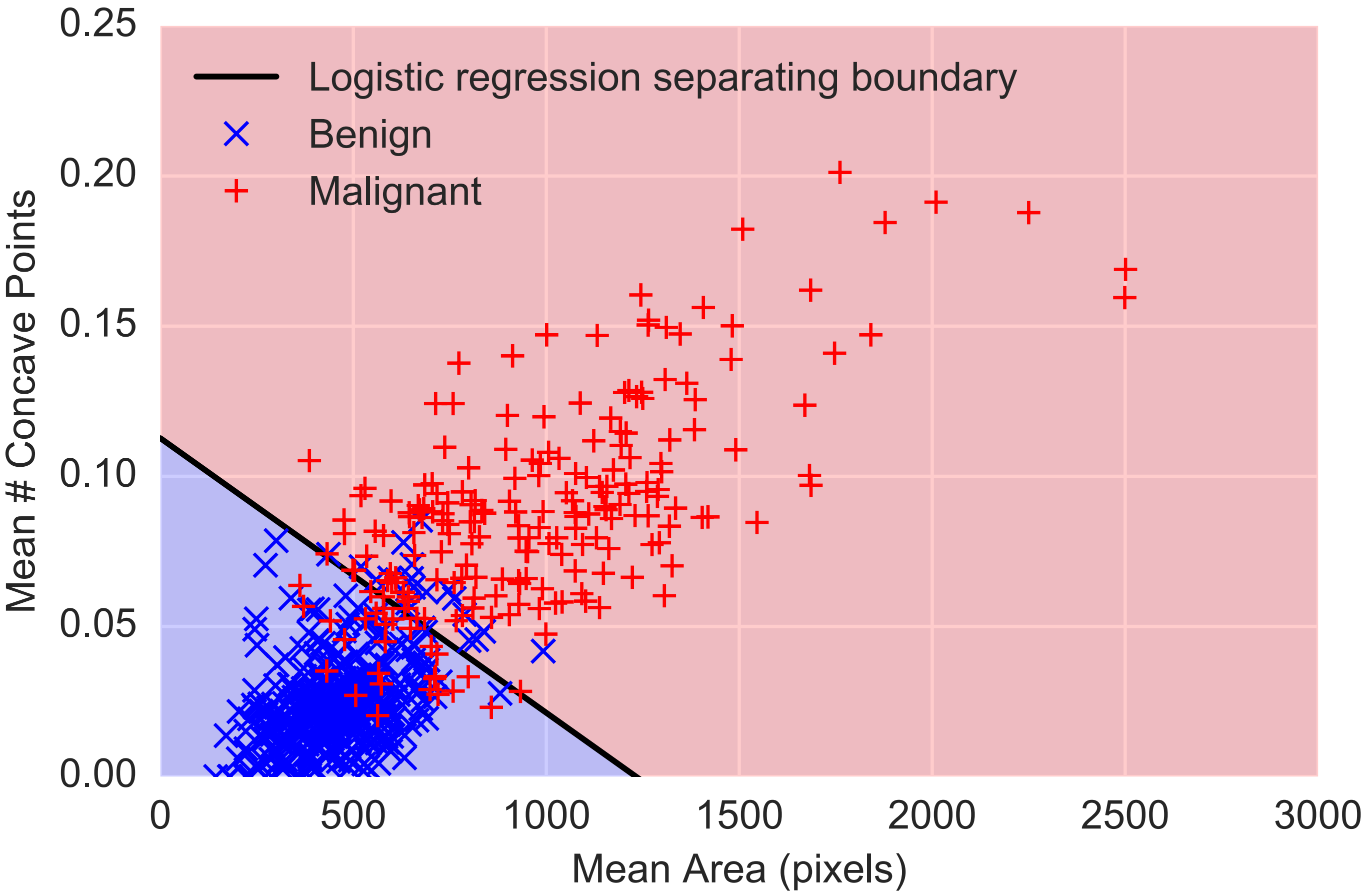
$$\operatorname*{minimize}_{\theta} \sum_{i=1}^{m} \log(1 + \exp(-y^{(i)} \cdot \theta^T x^{(i)}))$$

Gradient descent updates (can you derive these?):

$$\theta := \theta - \alpha \sum_{i=1}^{m} -y^{(i)} x^{(i)} \frac{1}{1 + \exp(y^{(i)} \cdot \theta^T x^{(i)})}$$

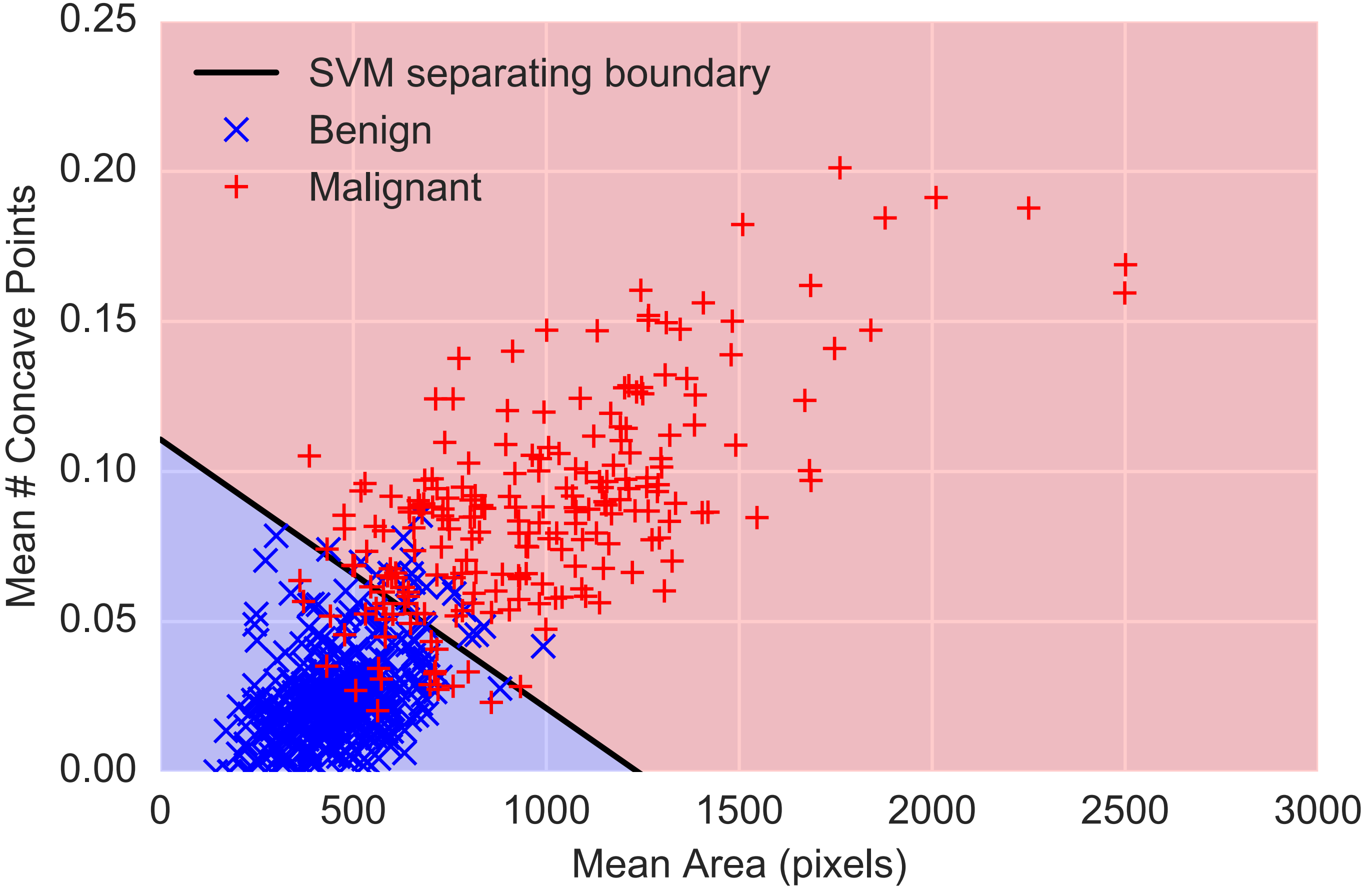
Logistic regression example

Running logistic regression on cancer data set, small regularization



Logistic regression example

Running logistic regression on cancer data set, small regularization



Multiclass classification

When output is in $\{1, \ldots, k\}$ (e.g., digit classification), a few different approaches

Approach 1: Build k different binary classifiers h_{θ_i} with the goal of predicting class i vs all others, output predictions as

$$\hat{y} = \operatorname*{argmax}_i h_{\theta_i}(x)$$

Approach 2: Use a hypothesis function $h_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^k$, define an alternative loss function $\ell : \mathbb{R}^k \times \{1, \ldots, k\} \rightarrow \mathbb{R}_+$

E.g., softmax loss (also called cross entropy loss):

$$\ell(h_\theta(x), y) = \log \sum_{j=1}^{k} \exp(h_\theta(x)_j) - h_\theta(x)_y$$

Outline

Example: classifying tumors

Classification in machine learning

Example classification algorithms

Classification with Python libraries

Support vector machine in scikit-learn

Train a support vector machine:

```python
from sklearn.svm import LinearSVC, SVC

clf = SVC(C=1e4, kernel='linear') # or
clf = LinearSVC(C=1e4, loss='hinge', max_iter=1e5)
clf.fit(X, y) # don’t include constant features in X
```

Make predictions:

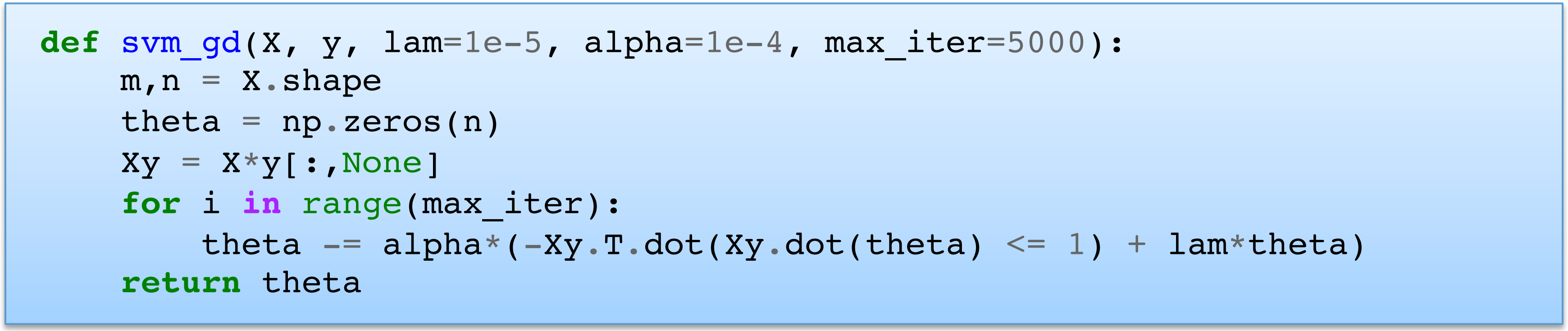
```python
y_pred = clf.predict(X)
```

Note: Scikit-learn in solving the problem (inverted regularization term):

$$\text{minimize}_\theta \quad C\sum_{i=1}^{m}\max\{1-y^{(i)}\cdot\theta^T x^{(i)},0\}+\frac{1}{2}\|\theta\|_2^2$$

Native Python SVM

It’s pretty easy to write a gradient-descent-based SVM too



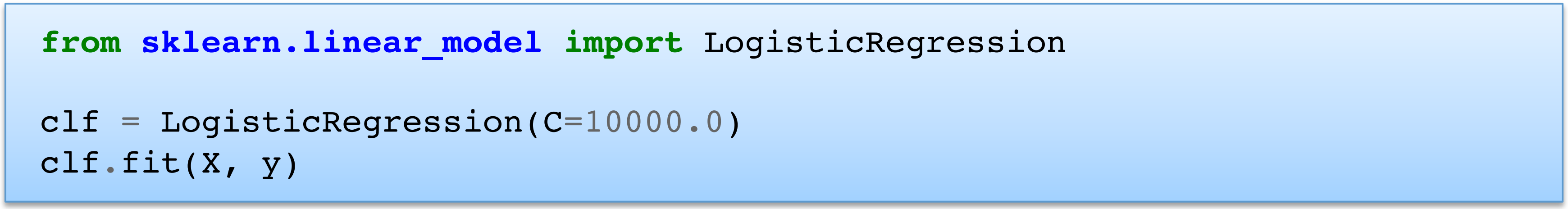
```python
def svm_gd(X, y, lam=1e-5, alpha=1e-4, max_iter=5000):
    m,n = X.shape
    theta = np.zeros(n)
    Xy = X*y[:,None]
    for i in range(max_iter):
        theta -= alpha*(-Xy.T.dot(Xy.dot(theta) <= 1) + lam*theta)
    return theta
```

For the most part, ML algorithms are very simple, you can easily write them yourself, but it’s fine to use libraries to quickly try many algorithms

But watch out for idiosyncratic differences (e.g., C vs λ, the fact that I’m using $y \in \{-1, +1\}$, not $y \in \{0,1\}$, etc)

Logistic regression in scikit-learn

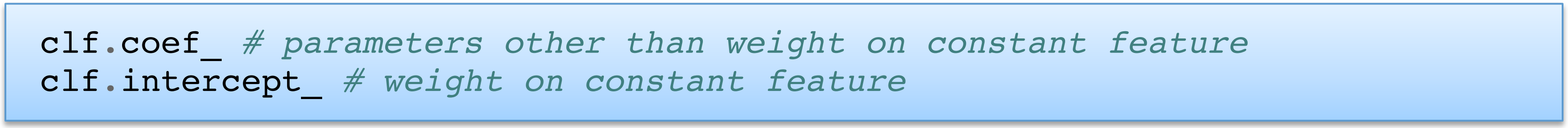
Admittedly very nice element of scikit-learn is that we can easily try out other algorithms



```python
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C=10000.0)
clf.fit(X, y)
```

For both this example and SVM, you can access resulting parameters using the fields



```python
clf.coef_ # parameters other than weight on constant feature
clf.intercept_ # weight on constant feature
```