Data Science

Week 8

Fitting a Model to Data - Classification (1):

SVM and Logistic Regression

Review for Week 6

1. Models and Prediction
2. Supervised Segmentation - Information Gain
3. Supervised Segmentation - Decision Tree
4. Probability Estimation
5. Example: Addressing the Churn Problem with Tree Induction

Formula of Entropy

Entropy measures the amount of information in a random variable

for binary classification

$$H(X) = -p_0 \log_2 p_0 - p_1 \log_2 p_1 \ , \ X = \{1, 0\}$$

Each p_i is the probability (the relative percentage) of property *i* within the set, ranging from $p_i = 1$ when all members of the set have property *i*, and $p_i = 0$ when no members of the set have property *i*.

for classification in *c* classes

$$H(X) = -\sum_{i=1}^{c} p_i \log_2 p_i = \sum_{i=1}^{c} p_i \log_2 1/p_i \ , \ X = \{1, \ldots, c\}$$

Calculation of the IG

- Notably, the entropy for each child (c_i) is weighted by the proportion of instances belonging to that child, $p(c_i)$.
- This addresses directly our concern from above that splitting off a single example, and noticing that that set is pure, may not be as good as splitting the parent set into two nice large, relatively pure subsets, even if neither is pure.
- As an example, consider the split in Figure. This is a two-class problem (• and ★). Examining the figure, the children sets certainly seem "purer" than the parent set. The parent set has 30 instances consisting of 16 dots and 14 stars, so:

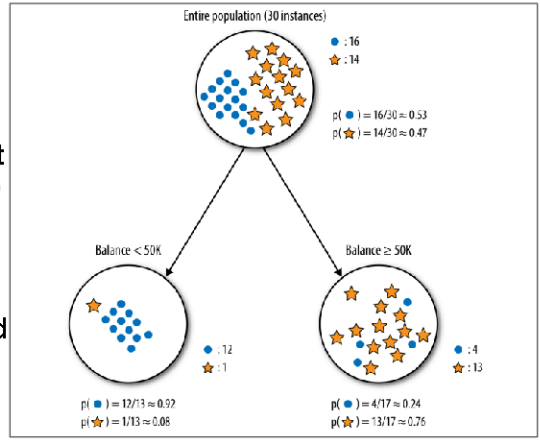


Figure 3-4. Splitting the "write-off" sample into two segments, based on splitting the Balance attribute (account balance) at 50K.

Calculation of the IG

entropy(parent) = - p(•) × $\log_2$ p(•) - p(☆) × $\log_2$ p(☆)

≈ - 0.53 × - 0.9 - 0.47 × - 1.1 ≈ 0.99 (very impure)

The entropy of the left child is:

entropy(Balance < 50K) = - p(•) × $\log_2$ p(•) - p(☆) × $\log_2$ p(☆)

≈ - 0.92 × (- 0.12) - 0.08 × (- 3.7) ≈ 0.39

The entropy of the right child is:

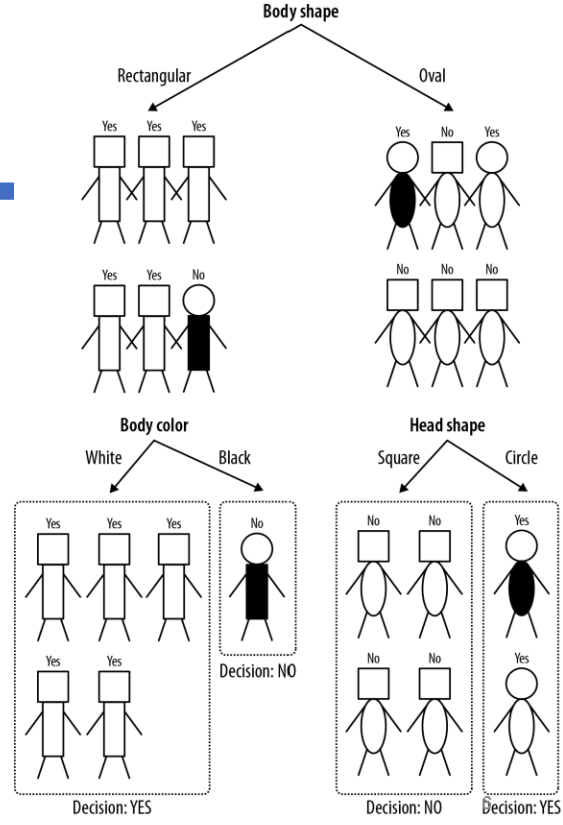
entropy(Balance ≥ 50K) = - p(•) × $\log_2$ p(•) - p(☆) × $\log_2$ p(☆)

≈ - 0.24 × (- 2.1) - 0.76 × (- 0.39) ≈ 0.79

IG = entropy(parent) - p(Balance < 50K) × entropy(Balance < 50K)
- p(Balance ≥ 50K) × entropy(Balance ≥ 50K)
≈ 0.99 - 13/30 × 0.39 - 17/30 × 0.79 ≈ 0.37

This split (Balance ≥ 50K or <50K)reduces entropy substantially. In predictive modeling terms, the attribute provides a lot of information on the value of the target.

Decision Tree



Outline

1. Regression via Mathematical Functions
2. Support Vector Machines, Briefly
3. Class Probability Estimation and Logistic "Regression"

Predictive Modeling- Supervised Segmentation

- As we have seen, predictive modeling involves finding a model of the target variable in terms of other descriptive attributes.
- We constructed a supervised segmentation model by recursively finding informative attributes on ever-more-precise subsets of the set of all instances, or from the geometric perspective, ever-more-precise subregions of the instance space.

Predictive Modeling- : Parametric Learning (1)

- An alternative method for learning a predictive model from a dataset is to start by specifying the structure of the model with certain numeric parameters left unspecified.
- Then the data mining calculates the best parameter values given a particular set of training data.
- A very common case is where the structure of the model is a parameterized mathematical function or equation of a set of numeric attributes.

Predictive Modeling- : Parametric Learning (2)

- The attributes used in the model could be chosen based on domain knowledge regarding which attributes ought to be informative in predicting the target variable, or they could be chosen based on other data mining techniques, such as the attribute selection procedures.
- The goal of the data mining is to tune the parameters so that the model fits the data as well as possible. This general approach is called parameter learning or parametric modeling.

Classification Tree using Entropy Measure

- Recall the instance-space view of tree models. One such diagram is replicated in the right Figure.
- It shows the space broken up into regions by horizontal and vertical decision boundaries that partition the instance space into similar regions.
- Examples in each region should have similar values for the target variable.
- We have seen how the entropy measure gives us a way of measuring homogeneity so we can choose such boundaries.

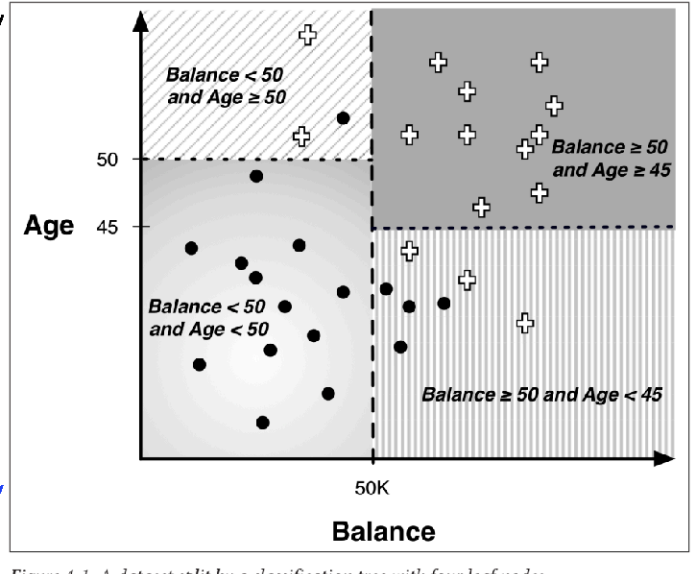


Figure 4-1. A dataset split by a classification tree with four leaf nodes.

Using Instance-space to Classify Unseen Instance

- A main purpose of creating homogeneous regions is so that we can predict the target variable of a new, unseen instance by determining which segment it falls into.
- For example, if a new customer falls into the lower-left segment, we can conclude that the target value is very likely to be "•" Similarly, if it falls into the upper-right segment, we can predict its value as "+".

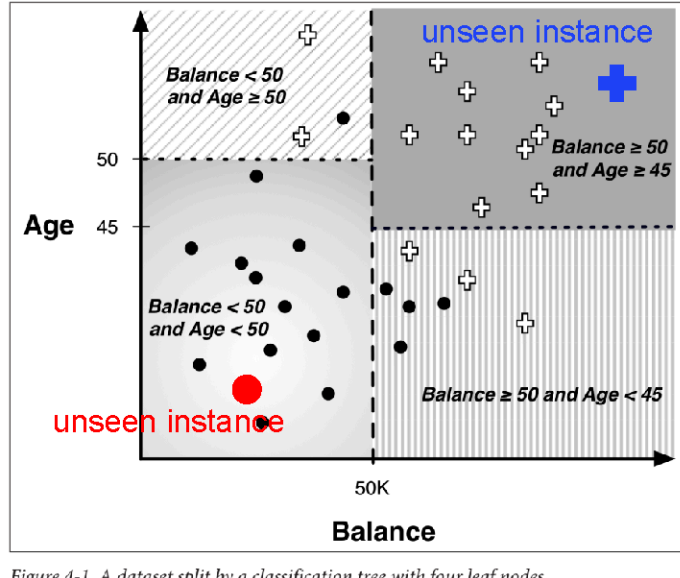
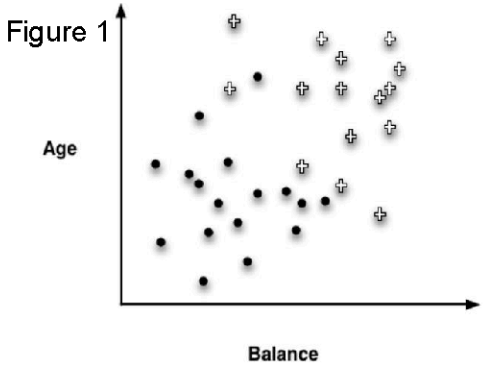


Figure 4-1. A dataset split by a classification tree with four leaf nodes.

Dataset with a Single Linear Split

- The instance-space view is helpful because if we take away the axis-parallel boundaries (see Figure 1) we can see that there clearly are other, possibly better, ways to partition the space.
- For example, we can separate the instances almost perfectly (by class) if we are allowed to introduce a boundary that is still a straight line, but is not perpendicular to the axes (Figure 2).

Figure 1



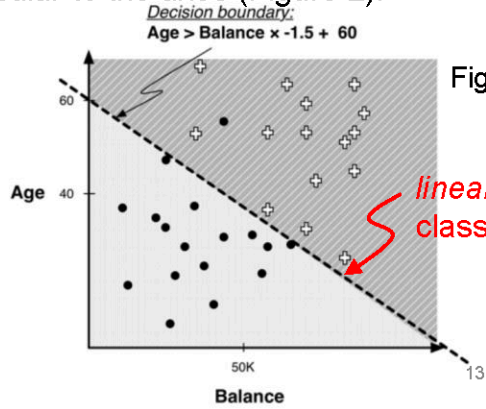


Figure 2

Linear Discriminant Functions

- Our goal is going to be to fit our model to the data, and to do so it is quite helpful to represent the model mathematically.
- You may recall that the equation of a line in two dimensions is $y = mx + b$, where m is the slope of the line and b is the y intercept (the y value when $x = 0$).
- The line in Figure of the page 13 can be expressed in this form (with Balance in thousands) as:

$$Age = (-1.5) \times Balance + 60$$

Classification Function

- We would classify an instance x as a + if it is above the line, and as a • if it is below the line. Rearranging this mathematically leads to the function that is the basis of all the techniques discussed in this chapter.
- First, for this example form the classification solution is shown in the following Equation.

Classification function:

$$\text{class}(x) = \begin{cases} + & \text{If} \quad 1.0 \times Age - 1.5 \times Balance + 60 > 0 \\ \bullet & \text{If} \quad 1.0 \times Age - 1.5 \times Balance + 60 \leq 0 \end{cases}$$

From Classification Function to Linear Discriminant

- Classification function is called a *linear discriminant* because it discriminates between the classes, and the function of the decision boundary is a linear combination—a weighted sum—of the attributes.
- In the two dimensions of our example, the linear combination corresponds to a line. In three dimensions, the decision boundary is a plane, and in higher dimensions it is a *hyperplane*.

Linear Discriminant: Weighted Sum of Attributes

- For our purposes, the important thing is that we can express the model as a weighted sum of the attribute values.
- Thus, this linear model is a different sort of multivariate supervised segmentation. Our goal with supervised segmentation still is to separate the data into regions with different values of the target variable. The difference is that the method for taking multiple attributes into account is to create a mathematical function of them.

General Linear Model

- In "Trees as Sets of Rules", we showed how a classification tree corresponds to a rule set—a logical classification model of the data. A linear discriminant function is a numeric classification model.
- For example, consider our feature vector **x**, with the individual component features being x_i. A linear model then can be written as follows in Equation:

Equation of a general linear model

$$f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots$$

- The example on page 14-15 can be written as:

$$f(\mathbf{x}) = 60 + 1.0 \times Age - 1.5 \times Balance$$

How to use and obtain the linear discriminant?

How to use the linear discriminant?

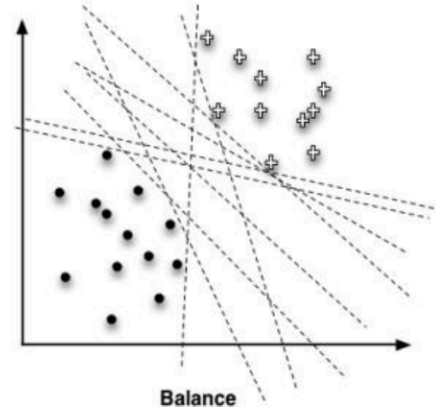
- To use this model as a linear discriminant, for a given instance represented by a feature vector **x**, we check whether $f(\mathbf{x})$ is positive or negative. As discussed above, in the two-dimensional case, this corresponds to seeing whether the instance **x** falls above or below the line.

How to obtain the linear discriminant?

- We now have a *parameterized* model: the weights of the linear function (w_i) are the parameters. The data mining is going to "fit" this parameterized model to a particular dataset—meaning specifically, to find a good set of weights on the features.

Many Different Possible Linear Boundaries for Classification

- After learning, these weights are often loosely interpreted as importance indicators of the features. Roughly, the larger the magnitude of a feature's weight, the more important that feature is for classifying the target.
- Unfortunately, it's not trivial to choose the "best" line to separate the classes. Let's consider a simple case, illustrated in the right Figure.
- Here the training data can indeed be separated by class using a linear discriminant. There actually are many different linear discriminants that can separate the classes perfectly. They have very different slopes and intercepts, and each represents a different model of the data.



Optimizing an Objective Function

- This brings us to one of the most important fundamental ideas in data mining—one that surprisingly is often overlooked even by data scientists themselves: we need to ask, what should be our goal or objective in choosing the parameters?
- In our case, this would allow us to answer the question: what weights should we choose? Our general procedure will be to define an objective function that represents our goal, and can be calculated for a particular set of weights and a particular set of data. We will then find the optimal value for the weights by maximizing or minimizing the objective function.

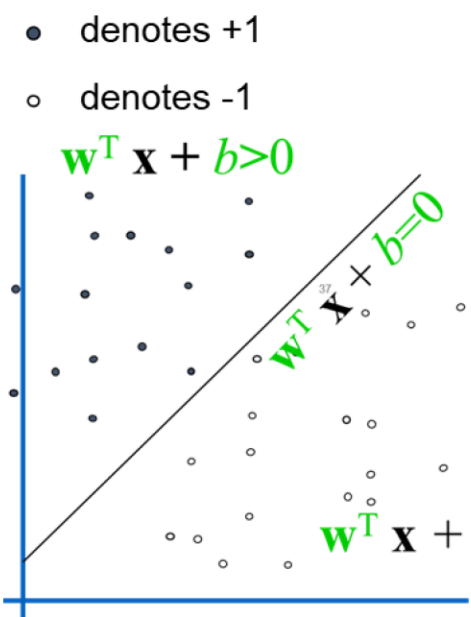
Two Choices for Binary Classification: SVM and Logistic Regression

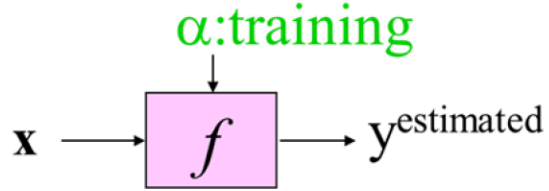
- Unfortunately, creating an objective function that matches the true goal (the above linear discriminant) of the data mining is usually impossible, so data scientists often choose based on faith and experience.
- Several choices have been shown to be remarkably effective. One of these choices creates the so-called "*support vector machine (SVM)*," about which we will say a few words after presenting a concrete example with a simpler objective function. Another choice is one of the most useful data mining techniques of all: *logistic regression*.

Outline

1. Regression via Mathematical Functions
2. Support Vector Machines, detail…
3. Class Probability Estimation and Logistic "Regression"

Linear Classifiers





$$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}^T \mathbf{x} + b)$$

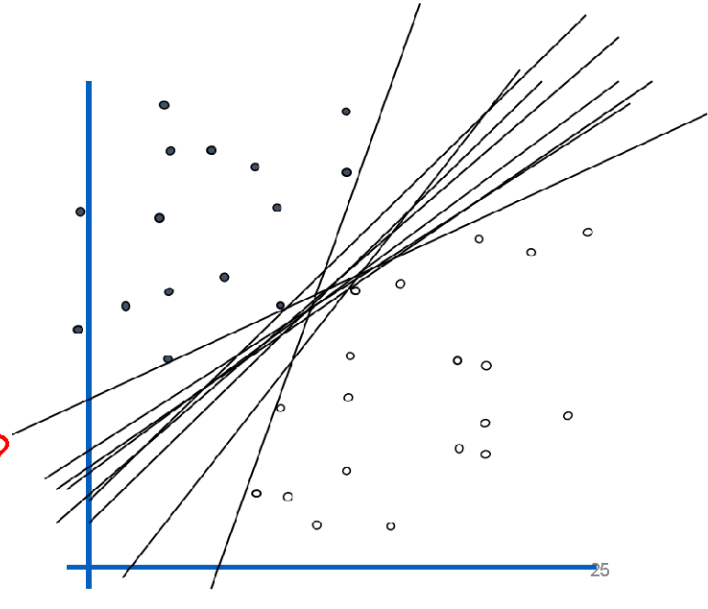
How would you classify this data?

Linear Classifiers

- denotes +1
- denotes -1

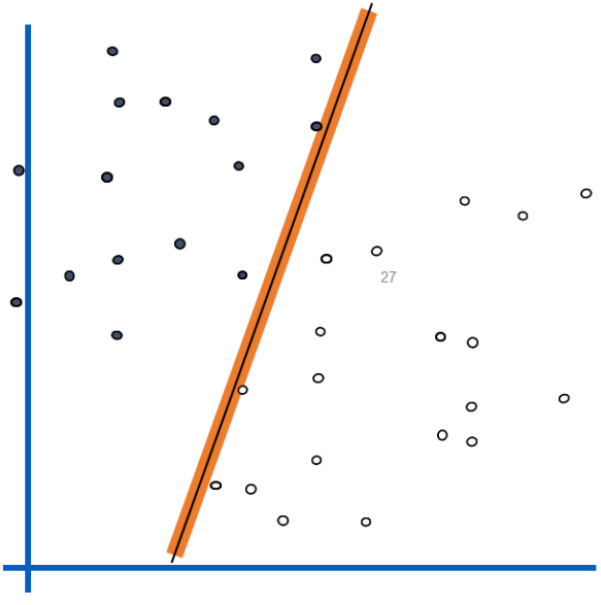
Any of these lines would be fine..

..but which is best?



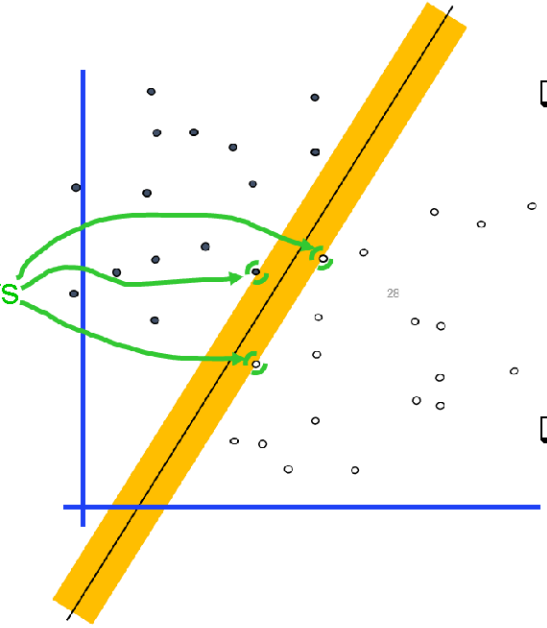
Classifier Margin

- Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.



Maximum Margin

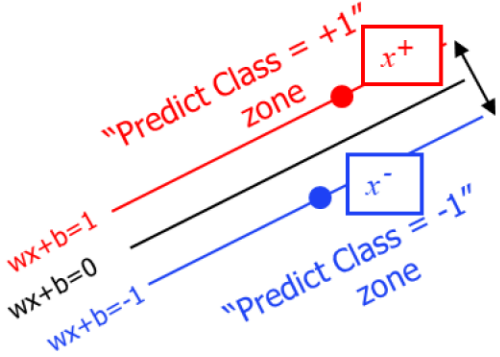
- denotes +1
- denotes -1



Support Vectors are those datapoints that the margin pushes up against

- The maximum margin linear classifier is the linear classifier with the maximum margin.
- This is the simplest kind of SVM

Linear SVM Mathematically



M =Margin Width

What we know:

$\mathbf{w}^T \mathbf{x}^+ + b = +1$

$\mathbf{w}^T \mathbf{x}^- + b = -1$

$\mathbf{w}^T (\mathbf{x}^+ - \mathbf{x}^-) = 2$

What we want to maximize:

$$M = \frac{(\mathbf{x}^+ - \mathbf{x}^-) \cdot \mathbf{w}}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$

Linear SVM Mathematically

- Goal: 1) Correctly classify all training data

$$\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b \geq 1 \quad \text{if } y_i = +1,$$
$$\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b \leq 1 \quad \text{if } y_i = -1$$
$$y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1 \quad \text{for all i}$$

2) Maximize the Margin $M = \frac{2}{\|\mathbf{w}\|}$

same as minimize $\frac{1}{2}\mathbf{w}^T\mathbf{w}$

Linear SVM Mathematically

- We can formulate a Quadratic Optimization Problem and solve for w and b
- Minimize $\frac{1}{2}\mathbf{w}^T\mathbf{w}$
- Subject to $y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1 \quad \forall i$

Optimization Problem (Optional)

Minimize $\| \mathbf{w} \| = \langle \mathbf{w} \cdot \mathbf{w} \rangle$ subject to $y_i(\langle \mathbf{x}_i \cdot \mathbf{w} \rangle + b) \geq 1$ for all i

Lagrangian method

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \| \mathbf{w} \| - \sum_i \alpha_i \left[\left(y_i (\mathbf{x}_i \cdot \mathbf{w}) + b \right) - 1 \right]$$

At the extremum, the partial derivative of L with respect both $\mathbf{w}$ and b must be 0. Taking the derivatives, setting them to 0, substituting back into L, and simplifying yields :

$$\text{Maximize } \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle$$

$$\text{subject to } \sum_i y_i \alpha_i = 0 \text{ and } \alpha_i \geq 0$$

Solution of the Optimization Problem (Optional)

- The solution has the form:

$$\mathbf{w} = \sum \alpha_i y_i \mathbf{x_i} \qquad b = y_k - \mathbf{w}^{\mathbf{T}} \mathbf{x}_k \text{ for any } \mathbf{x}_k \text{ such that } \alpha_k \neq 0$$

- Each non-zero α_i indicates that corresponding $\mathbf{x_i}$ is a support vector.
- Then the classifying function will have the form:

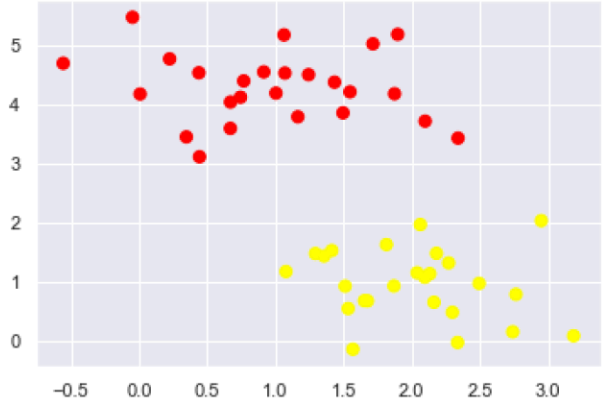
$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^{\mathbf{T}} \mathbf{x} + b$$

SMO (sequential minimal optimization) can be used to solve the optimization problem

http://cs229.stanford.edu/notes/cs229-notes3.pdf

Data for SVM

- Consider the simple case of a classification task, in which the two classes of points are well separated



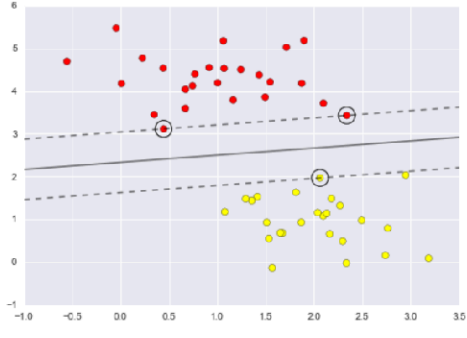
SVM in scikit learn

- Let's see the result of an actual fit to this data: we will use Scikit-Learn's support vector classifier to train an SVM model on this data.

```
62 import numpy as np
63 import matplotlib.pyplot as plt
64 from scipy import stats
65 import seaborn as sns; sns.set()
66 from sklearn.datasets.samples_generator import make_blobs
67 X, y = make_blobs(n_samples=50, centers=2,
68 random_state=0, cluster_std=0.60)
69 plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');
70
71 from sklearn.svm import SVC # "Support vector classifier"
72 model = SVC(kernel='linear', C=1E10)
73 model.fit(X, y)
```

Visualization of LSVM model

```
75 def plot_svc_decision_function(model, ax=None, plot_support=True):
76     """Plot the decision function for a two-dimensional SVC"""
77     if ax is None:
78         ax = plt.gca()
79     xlim = ax.get_xlim()
80     ylim = ax.get_ylim()
81     # create grid to evaluate model
82     x = np.linspace(xlim[0], xlim[1], 30)
83     y = np.linspace(ylim[0], ylim[1], 30)
84     Y, X = np.meshgrid(y, x)
85     xy = np.vstack([X.ravel(), Y.ravel()]).T
86     P = model.decision_function(xy).reshape(X.shape)
87     # plot decision boundary and margins
88     ax.contour(X, Y, P, colors='k',
89     levels=[-1, 0, 1], alpha=0.5,
90     linestyles=['--', '-', '--'])
91     # plot support vectors
92     if plot_support:
93         ax.scatter(model.support_vectors_[:, 0],
94         model.support_vectors_[:, 1],
95         s=300, linewidth=1, facecolors='none');
96     ax.set_xlim(xlim)
97     ax.set_ylim(ylim)
98 #
99 plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
100 plot_svc_decision_function(model);
```



Outline

1. Regression via Mathematical Functions
2. Support Vector Machines, Briefly
3. Class Probability Estimation and Logistic "Regression"

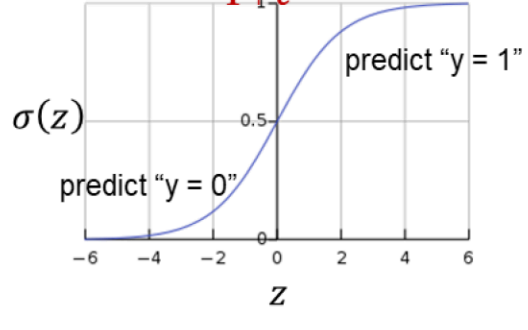
Logistic Regression

Given $\mathbf{x}, y$, want $\hat{y} = P(y = 1|\mathbf{x})$ infinitely close to y,

Where, $\mathbf{x} \in \mathbb{R}^{n_x}, 0 \leq \hat{y} \leq 1$, output predicted value $\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b)$

Unknown Parameters: $\mathbf{w} \in \mathbb{R}^{n_x}, b \in \mathbb{R}$

Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$



$$z \to +\infty, \sigma(z) \approx \frac{1}{1+e^{-\infty}} = 1$$

$$z \to -\infty, \sigma(z) \approx \frac{1}{1+e^{+\infty}} = 0$$

Logistic Regression

Given $\mathbf{x}$, want $\hat{y} = P(y = 1|\mathbf{x})$→y

Where, $\mathbf{x} \in \mathbb{R}^{n_x}, 0 \leq \hat{y} \leq 1$

Output $\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b)$

Parameters: $\mathbf{w} \in \mathbb{R}^{n_x}, b \in \mathbb{R}$

$$\boldsymbol{x_0} = 1, \mathbf{x} \in \mathbb{R}^{n_x+1}$$

$$\hat{y} = \sigma(\boldsymbol{\theta}^T\mathbf{x})$$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix}$$

Suppose: predict "y = 1" if $\sigma(\boldsymbol{\theta}^T\mathbf{x}) \geq 0.5$

predict "y = 0" if $\sigma(\boldsymbol{\theta}^T\mathbf{x}) < 0.5$

Logistic Regression Loss Function

Logistic Regression $\hat{y}^{(i)} = \sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right), 0 \leq \hat{y}^{(i)} \leq 1, \quad \sigma(z) = \frac{1}{1+e^{-z}}$

Training set with m samples

$$\left(\mathbf{x}^{(1)}, y^{(1)}\right), \left(\mathbf{x}^{(2)}, y^{(2)}\right), \cdots, \left(\mathbf{x}^{(m)}, y^{(m)}\right)$$

$$\mathbf{x}^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}, x_0^{(i)} = 1, y^{(i)} \in \{0,1\}$$

How to choose parameters $\boldsymbol{\theta}$ to make $\hat{y}^{(i)}$ infinitely close to $y^{(i)}$?

Logistic Regression Loss Function

Loss (error) function:

$$L(\hat{y}^{(i)}, y^{(i)}) = -y^{(i)}\ln(\hat{y}^{(i)}) - (1 - y^{(i)})\ln(1 - \hat{y}^{(i)})$$

$L(\hat{y}^{(i)}, y^{(i)}) \geq 0$

if $y^{(i)} = 1, (1 - y^{(i)}) = 0$

when $L(\hat{y}^{(i)}, y^{(i)}) = -\ln(\hat{y}^{(i)}) \rightarrow 0, \hat{y}^{(i)} \rightarrow 1$

if $y^{(i)} = 0, y^{(i)} = 0$

when $L(\hat{y}^{(i)}, y^{(i)}) = -\ln(1 - \hat{y}^{(i)}) \rightarrow 0, \hat{y}^{(i)} \rightarrow 0$

Logistic Regression

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$
$$= -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \ln\left(\hat{y}^{(i)}\right) + (1 - y^{(i)}) \ln\left(1 - \hat{y}^{(i)}\right)\right]$$

Learning: find parameter $\boldsymbol{\theta}$ to $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

Prediction: given new x output $\hat{y} = \sigma(\boldsymbol{\theta}^T \mathbf{x}) = \frac{1}{1+e^{-\boldsymbol{\theta}^T \mathbf{x}}}$

Gradient Descent

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \ln\left(h_{\boldsymbol{\theta}}\left(\mathrm{x}^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \ln\left(1 - h_{\boldsymbol{\theta}}\left(\mathrm{x}^{(i)}\right)\right)\right]$$

$$where, h_{\boldsymbol{\theta}}\left(\mathrm{x}^{(i)}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \mathrm{x}^{(i)}}}$$

Goal: $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

Good news: $J(\boldsymbol{\theta})$ is a Convex function!

Bad news: No analytical solution

Convex function

Non-Convex function

http://cs230.stanford.edu/files/C1M2.pdf

Gradient Descent

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \ln\left(h_{\boldsymbol{\theta}}\left(x^{(i)}\right)\right) + (1-y^{(i)}) \ln\left(1-h_{\boldsymbol{\theta}}\left(x^{(i)}\right)\right)\right]$$

Goal: $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

Good news: Convex function!

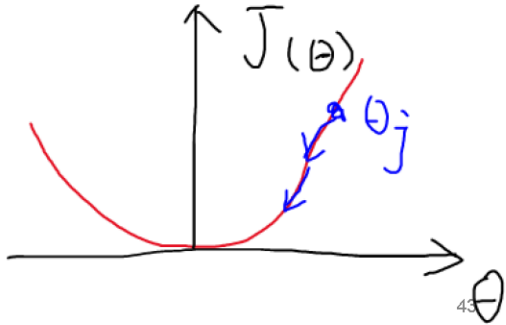
Bad news: No analytical solution

Repeat

{

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

}



Gradient Descent

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \ln\left(h_{\boldsymbol{\theta}}\left(x^{(i)}\right)\right) + (1-y^{(i)}) \ln\left(1-h_{\boldsymbol{\theta}}\left(x^{(i)}\right)\right)\right]$$

Goal: $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

Good news: Convex function!
Bad news: No analytical solution

Repeat

{

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

}

(Simultaneously update all θ_j)

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

Remember !

$$\frac{d}{dx} ln\big(\mathrm{f}(x)\big) = \frac{1}{f(x)} * f'$$

$$\frac{d}{dx} e^{-x} = -e^{-x}$$

$$\frac{d\left(\frac{f(x)}{g(x)}\right)}{dx} = \frac{(f'g - g'f)}{g^2}$$

The derivative of the sigmoid function is

$$\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \left(\frac{1}{1+e^{-x}}\right)\left(\frac{e^{-x}}{1+e^{-x}}\right) \\
&= \left(\frac{1}{1+e^{-x}}\right)\left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) \\
&= \sigma(x)\left(\frac{1+e^{-x}}{1+e^{-x}} - \sigma(x)\right) \\
&= \sigma(x)\left(1-\sigma(x)\right)
\end{aligned}$$

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \ \ln h_\theta(x^{(i)}) + (1 - y^{(i)}) \ \ln(1 - h_\theta(x^{(i)}))]$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{\partial}{\partial \theta_j} \frac{-1}{m} \sum_{i=1}^{m} [y^{(i)} \ ln(h_\theta(x^{(i)}) + (1 - y^{(i)}) \ ln(1 - h_\theta(x^{(i)}))]$$

$$\underset{\text{linearity}}{=} \frac{-1}{m} \sum_{i=1} [y^{(i)} \frac{\partial}{\partial \theta_j} \ ln(h_\theta(x^{(i)})) + (1 - y^{(i)}) \frac{\partial}{\partial \theta_j} \ ln(1 - h_\theta(x^{(i)}))]$$

$$\underset{\text{chain rule}}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \frac{\frac{\partial}{\partial \theta_j} \ h_\theta(x^{(i)})}{h_\theta(x^{(i)})} + (1 - y^{(i)}) \frac{\frac{\partial}{\partial \theta_j}\left(1 - h_\theta(x^{(i)})\right)}{1 - h_\theta(x^{(i)})} \right]$$

$$\frac{d}{dx} ln(\mathrm{f}(x)) = \frac{1}{f(x)} * f'$$

$$\underset{h_\theta(x) = \sigma(\theta^\top x)}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \frac{\frac{\partial}{\partial \theta_j} \sigma(\theta^\top x^{(i)})}{h_\theta(x^{(i)})} + (1 - y^{(i)}) \frac{\frac{\partial}{\partial \theta_j}\left(1 - \sigma(\theta^\top x^{(i)})\right)}{1 - h_\theta(x^{(i)})} \right]$$

Derivative of sigmoid

$$\underset{\sigma'}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \frac{\sigma(\theta^\top x^{(i)})\left(1 - \sigma(\theta^\top x^{(i)})\right) \frac{\partial}{\partial \theta_j}(\theta^\top x^{(i)})}{h_\theta(x^{(i)})} - (1 - y^{(i)}) \frac{\sigma(\theta^\top x^{(i)})\left(1 - \sigma(\theta^\top x^{(i)})\right) \frac{\partial}{\partial \theta_j}(\theta^\top x^{(i)})}{1 - h_\theta(x^{(i)})} \right]$$

$$\underset{\sigma'}{=} \frac{-1}{m}\sum_{i=1}^{m}\left[y^{(i)}\frac{\sigma(\theta^\top x^{(i)})\left(1-\sigma(\theta^\top x^{(i)})\right)\frac{\partial}{\partial\theta_j}(\theta^\top x^{(i)})}{h_\theta(x^{(i)})} - (1-y^{(i)})\frac{\sigma(\theta^\top x^{(i)})\left(1-\sigma(\theta^\top x^{(i)})\right)\frac{\partial}{\partial\theta_j}(\theta^\top x^{(i)})}{1-h_\theta(x^{(i)})}\right]$$

$$\underset{\sigma(\theta^\top x)=h_\theta(x)}{=} \frac{-1}{m}\sum_{i=1}^{m}\left[y^{(i)}\frac{h_\theta(x^{(i)})\left(1-h_\theta(x^{(i)})\right)\frac{\partial}{\partial\theta_j}(\theta^\top x^{(i)})}{h_\theta(x^{(i)})} - (1-y^{(i)})\frac{h_\theta(x^{(i)})\left(1-h_\theta(x^{(i)})\right)\frac{\partial}{\partial\theta_j}(\theta^\top x^{(i)})}{1-h_\theta(x^{(i)})}\right]$$

$$\underset{\frac{\partial}{\partial\theta_j}(\theta^\top x^{(i)})=x_j^{(i)}}{=} \frac{-1}{m}\sum_{i=1}^{m}\left[y^{(i)}\left(1-h_\theta(x^{(i)})\right)x_j^{(i)} - (1-y^i)h_\theta(x^{(i)})x_j^{(i)}\right]$$

$$\underset{\text{distribute}}{=} \frac{-1}{m}\sum_{i=1}^{m}\left[y^i - y^i h_\theta(x^{(i)}) - h_\theta(x^{(i)}) + y^{(i)}h_\theta(x^{(i)})\right]x_j^{(i)}$$

$$\underset{\text{cancel}}{=} \frac{-1}{m}\sum_{i=1}^{m}\left[y^{(i)} - h_\theta(x^{(i)})\right]x_j^{(i)}$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left[h_\theta(x^{(i)}) - y^{(i)}\right]x_j^{(i)}$$

Pseudocode of Logistic Regression

$w_1 = 0, w_2 = 0, b = 0, \alpha = 4$

k is the number of iterations
m is the number of sample data

for t = 1 to k:

$J = 0, dw_1 = 0, dw_2 = 0, db = 0$

for i = 1 to m:

$z^{(i)} = w_1 x_1^{(i)} + w_2 x_2^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -\left[y^{(i)} \ln a^{(i)} + (1 - y^{(i)}) \ln(1 - a^{(i)})\right]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$

$db \mathrel{+}= dz^{(i)}$

$J = J/m, dw_1 = dw_1/m, dw_2 = dw_2/m, db = db/m$

$w_1 = w_1 - \alpha\, dw_1, w_2 = w_2 - \alpha\, dw_2, b = b - \alpha\, db$

Gradient Descent

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \ \ln h_\theta(x^{(i)}) + (1 - y^{(i)}) \ \ln(1 - h_\theta(x^{(i)}))]$$

Want : $\min_\theta J(\theta)$

Repeat{ $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$ } (simultaneously update all θ_j)

Repeat{ $\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$ } (simultaneously update all θ_j)

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1} \text{ and } x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

Logistic regression in scikit learn

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
X, y = load_iris(return_X_y=True)

clf = LogisticRegression(solver='lbfgs', multi_class='auto',
                         random_state=0, max_iter=1200).fit(X, y)
print(clf.predict(X[:2, :]))
print(clf.predict_proba(X[:2, :]))
print(clf.score(X, y))
```

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Summary

1. Regression via Mathematical Functions
2. Support Vector Machines, Briefly
3. Class Probability Estimation and Logistic "Regression"

Reference

- Provost, Foster, and Tom Fawcett. Data Science for Business: What you need to know about data mining and data-analytic thinking. O'Reilly Media, Inc., 2013.

- VanderPlas, J. (2016). Python data science handbook: Essential tools for working with data. " O'Reilly Media, Inc.".

Data Science

Week 9

Fitting a Model to Data (2)-Classification:

Neural Networks