CAI 4104/6108 – Machine Learning Engineering:

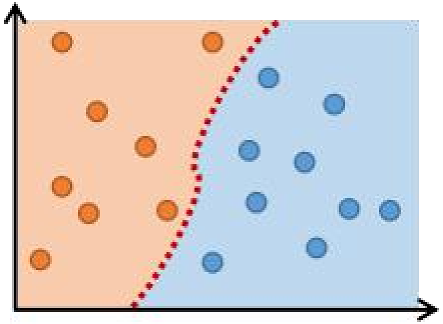
Logistic Regression & SVM

Prof. Vincent Bindschaedler

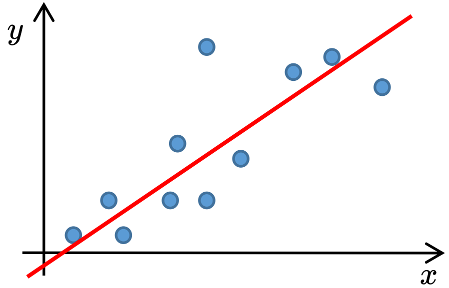
Spring 2024

Reminder: Supervised Learning

- Classification
  - Task: predict the corresponding label
  - Different types:
    - Binary classification: there are only two classes (0,1; +,-, etc.)
    - Multiclass: more than two classes
    - Multi-label: each instance can belong to more than one class
    - One-class: there is only one class, we want to distinguish it from everything else



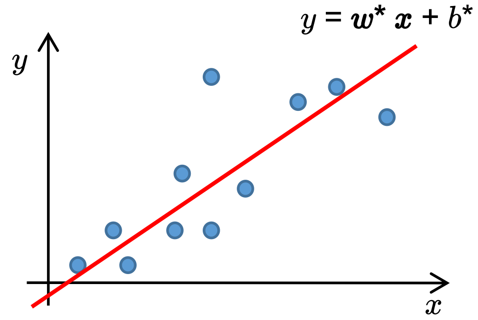
- Regression
  - Task: predict the corresponding value (typically a real number) or target
    - E.g.: you want to predict a person's future income based on their high school GPA



- Others:
  - Sequence-to-sequence, similarity learning/metric learning, learning to rank, etc.

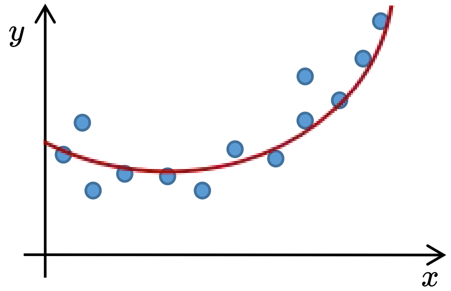
Reminder: Linear Regression

- Dataset
  - Matrix $\mathbf{X}$ $(n \times m)$ and the target vector $\mathbf{y}$ $(n \times 1)$
    - Let $\boldsymbol{x}_i$ be the feature vector for example i and $y_i \in \mathbb{R}$ is the corresponding target/value
- Prediction task:
  - Given a feature vector $\boldsymbol{x}$, predict the target/value $y \in \mathbb{R}$ as accurately as possible
- Linear Regression:
  - The model is: $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = h_{\boldsymbol{w},b}(\boldsymbol{x}) = \boldsymbol{w}\ \boldsymbol{x} + b$
  - The prediction is: $y = h_{\boldsymbol{\theta}}(\boldsymbol{x})$
- Training:
  - We want to minimize the Mean Squared Error (MSE) [this is called OLS]
    - $\text{MSE}(\boldsymbol{w},b) := \text{MSE}(h_{\boldsymbol{w},b}, \boldsymbol{X}, \boldsymbol{y}) = 1/n \sum_i [h_{\boldsymbol{w},b}(\boldsymbol{x}_i) - y_i]^2 = 1/n \sum_i [\boldsymbol{w}\ \boldsymbol{x}_i + b - y_i]^2$
  - Optimal parameters: $\boldsymbol{\theta}^* = (\boldsymbol{w}^*, b^*) = \text{argmin}_{\boldsymbol{w},b}\ \text{MSE}(\boldsymbol{w},b)$
  - Remark: MSE is the *expected* squared error loss
    - Squared Error Loss (L_2 loss): $L(\boldsymbol{\theta}) = [y - h_{\boldsymbol{\theta}}(\boldsymbol{x})]^2$



Reminder: Polynomial Regression

- What if the data is non-linear?
  - Then a linear model won't fit (it will have high bias)
- Can we still use linear regression?
  - Yes, we can fit a linear model on non-linear data!
  - How? Add features that can capture non-linearity!
  - Example: suppose we have a single feature
    - The linear regression model is: $h_\theta(\boldsymbol{x}) = wx + b$
    - If we add x^2 as a feature, then the model is: $h_\theta(\boldsymbol{x}) = w_1 x + w_2 x^2 + b$

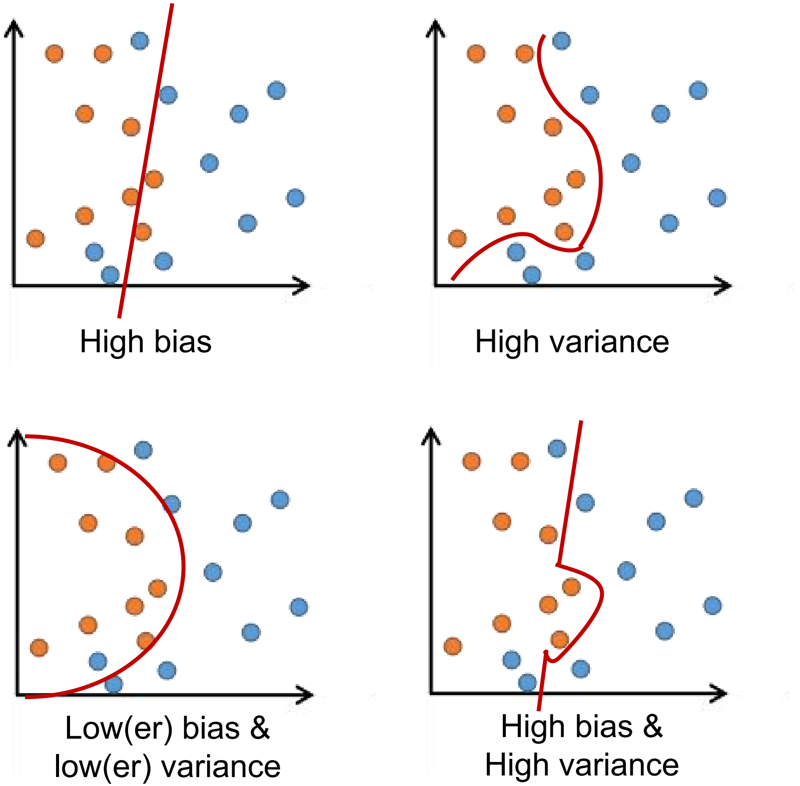


- Polynomial regression
  - If we have several features, say x, y, z, then we can consider all combinations of features up to some degree. That is:
    - x^3, y^3, z^3, x^2y, x^2z, y^2x, y^2z, z^2x, z^2y, xyz (and x, y, z, 1)
  - Q: If we have m features and want all combinations up to degree k, how many features do we get?
    - $m+k$ choose k: $C(m+k, k) = (m+k)! / (m!\, k!)$

Reminder: Bias and Variance

- Bias
  - Error due to incorrect assumptions in the model
  - *Inability to capture the true relationship*
- Variance
  - Sensitivity to small variations in the training data

- Ideally, we want: low bias **and** low variance
  - Strategies to lower bias:
    - Increase model complexity
    - Use more features
  - Strategies to lower variance:
    - Reduce model complexity
    - Use more training data



Reminder: Bias-Variance Tradeoff

- Generalization error (aka out-of-sample error or risk)
  - Prediction error on *unseen* data
  - Related to overfitting
    - If the model overfits, then the generalization error will be large
- Bias-Variance Tradeoff
  - Generalization error: `bias`^{2} `+ variance + irreducible error`
    - For more details:
      - Geman et al. "*Neural networks and the bias/variance dilemma.*" Neural computation (1992)
      - Kohavi et al. "Bias plus variance decomposition for zero-one loss functions." ICML, 1996.
  - Why is it a tradeoff?
    - Increasing model complexity => lower bias
    - Decreasing model complexity => lower variance
    - Note: there has been some debate of whether this applies to neural networks
      - E.g.: see Neal et al. "A modern take on the bias-variance tradeoff in neural networks." arXiv, 2018.

Overfitting & Regularization

- Most models can be regularized
  - Typically tuned through a regularization constant (hyperparameter)
  - Effect: lower variance at the cost of higher bias

- Regularization reduces model complexity
  - It decreases the degrees of freedom of the model

- If your model is overfitted
  - Regularization is (one of) the first things you should try

- How?
  - Regularization linear regression: Minimize MSE($\boldsymbol{w}$,b) + λ $||\boldsymbol{w}||^2$
    - In other words: we are adding a penalty term with regularization constant λ to the loss function

Types of Regularization

- Suppose our loss function is $L(\boldsymbol{\theta})$ and let λ be our regularization constant
- L_1-regularization: minimize $J(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda||\boldsymbol{w}||_1 = L(\boldsymbol{\theta}) + \lambda\sum_i |w_i|$
  - Effect: encourage sparsity in the weights (i.e., weights of least important features will be close 0)
  - In the context of linear regression, this is also called Least Absolute Selection and Shrinkage Operator (LASSO) regression
- L_2-regularization: minimize $J(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda||\boldsymbol{w}||_2 = L(\boldsymbol{\theta}) + \lambda\sum_i |w_i|^2$
  - Effect: encourage minimization of the weights (i.e., weights will be close to 0)
  - In the context of linear regression, this is also called Tikhonov regularization or Ridge regression
- Elastic net regularization: minimize $J(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda_1||\boldsymbol{w}||_1 + \lambda_2||\boldsymbol{w}||_2$
  - If we let $\alpha=\lambda_1/(\lambda_1+\lambda_2)$, then if $\alpha=0$, we get L_2 regularization; if $\alpha=1$, we get L_1 regularization
- L_0-regularization: minimize $J(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda||\boldsymbol{w}||_0$
  - Effect: encourage as few non-zero entries in the weights vector as possible
    - Note: $||\boldsymbol{w}||_0=\sum_i 1(w_i)\neq 0$. In other words: $||\boldsymbol{w}||_0$ is the number of non-zero weights

(More) Types of Regularization

- Suppose our loss function is $L(\boldsymbol{\theta})$ and let λ be our regularization constant
- L$_1$-regularization: minimize $J(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda||\boldsymbol{w}||_1 = L(\boldsymbol{\theta}) + \lambda\sum_i |w_i|$
- L$_2$-regularization: minimize $J(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda||\boldsymbol{w}||_2 = L(\boldsymbol{\theta}) + \lambda\sum_i |w_i|^2$
- Elastic net regularization: minimize $J(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda_1||\boldsymbol{w}||_1 + \lambda_2||\boldsymbol{w}||_2$
- L$_0$-regularization: minimize $J(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda||\boldsymbol{w}||_0$

- Best practice:
  - *Rescale the data (min-max normalize or standardize) before regularizing!* **Why?**

- There are other approaches for regularization. For example:
  - Early stopping: stop when the validation error starts increasing
  - Dilution (aka Dropout) for neural networks

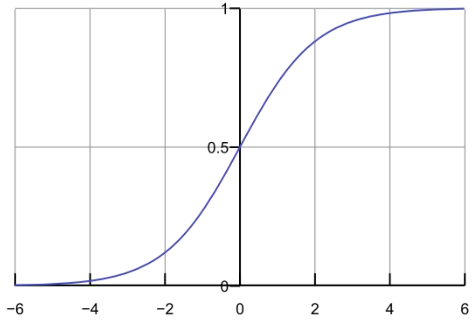
Logistic Regression

- Can we do binary classification with a linear regression model?
  - Or phrased differently: what does binary classification with a simple linear model look like?
- Logistic regression:
  - Setup: Let $\boldsymbol{x}_i$ be the feature vector for example i and $y_i \in \{0,1\}$ is the corresponding label
  - Note: it has *regression* in the name, but it is a *classification* model!
  - Idea: recast predicting the class label as predicting the probability of the class label
    - Informally: use a linear model to predict a score z_i for each example $\boldsymbol{x}_i$ such that the larger z_i is, the more likely it is that the label is 1 (or the smaller z_i is the more likely it is the label is 0)
  - The model is: $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = h_{\boldsymbol{w},b}(\boldsymbol{x}) = 1 / [1 + \exp\{-(\boldsymbol{w}\,\boldsymbol{x} + b)\}]$ [$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = 1/(1+e^{-z})$ where $z = \boldsymbol{w}\,\boldsymbol{x} + b$]
    - The function $f(z) = 1 / (1+e^{-z})$ is a link function; it is called the logistic function or sigmoid function
    - The logistic function is the inverse of the logit function: $\mathrm{logit}(p) = \log [\, p /(1-p) \,]$
  - Prediction: if $p \geq 0.5$, we predict 1 otherwise we predict 0. Here: $p = h_{\boldsymbol{\theta}}(\boldsymbol{x})$
    - We can interpret $p = h_{\boldsymbol{\theta}}(\boldsymbol{x})$ as the probability of label being 1 (and $1-p$ as the probability of label being 0)

Logistic Regression

- Logistic regression:
  - Setup: Let $\boldsymbol{x}_i$ be the feature vector for example i and $y_i \in \{0,1\}$ is the corresponding label
  - The model is: $h_\theta(\boldsymbol{x}) = h_{w,b}(\boldsymbol{x}) = 1 / [1 + \exp\{-(\boldsymbol{w}\,\boldsymbol{x} + b)\}]$ [$h_\theta(\boldsymbol{x}) = 1/(1+e^{-z})$ where $z = \boldsymbol{w}\,\boldsymbol{x} + b$]
    - The function $f(z) = 1 / (1+e^{-z})$ is a link function; it is called the logistic function or sigmoid function
    - The logistic function is the inverse of the logit function: $\mathrm{logit}(p) = \log [\, p /(1-p)\,]$
  - Prediction: Let $p = h_\theta(\boldsymbol{x})$
    - If $p \geq 0.5$, we predict 1 otherwise we predict 0
    - We can interpret $p = h_\theta(\boldsymbol{x})$ as the probability of label being 1 (and $1-p$ as the probability of label being 0)

- How do we train the model?
  - How do we find optimal parameters $\boldsymbol{w}^*$, b^*?
    - For examples with label 1, the probability p should be high
    - For examples with label 0, the probability p should be low



source: wikipedia

Logistic Regression

- Logistic regression:
  - Setup: Let $\boldsymbol{x}_i$ be the feature vector for example i and $y_i \in \{0,1\}$ is the corresponding label
  - The model is: $h_{\theta}(\boldsymbol{x}) = h_{w,b}(\boldsymbol{x}) = 1 / [1 + \exp\{-(\boldsymbol{w}\,\boldsymbol{x} + b)\}]$ [$h_{\theta}(\boldsymbol{x}) = 1/(1+e^{-z})$ where $z = \boldsymbol{w}\,\boldsymbol{x} + b$]
  - Prediction: Let $p = h_{\theta}(\boldsymbol{x})$
    - If $p \geq 0.5$, we predict 1 otherwise we predict 0

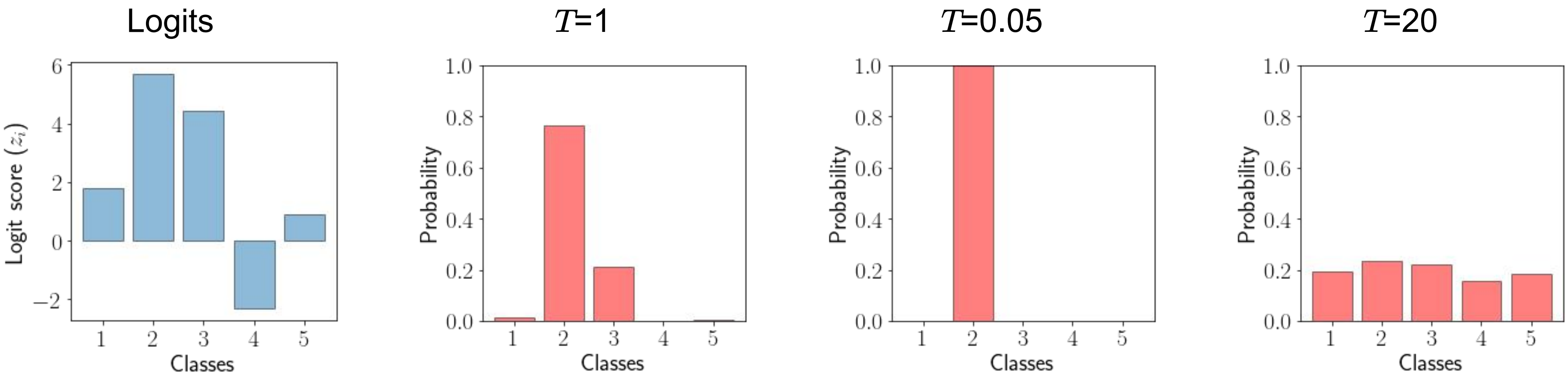
- Training: how do we find optimal parameters $\boldsymbol{w}^*$, b^*?
  - $L(\boldsymbol{\theta}) = -1/n \sum_i y_i \log(p_i) + (1-y_i) \log(1-p_i)$ where $p_i = h_{\theta}(\boldsymbol{x}_i)$
    - This is called the *logistic loss*, *binary cross-entropy* and also *log loss*
  - How do we minimize $L(\boldsymbol{\theta})$?
    - There is no closed form!
    - But we can use optimization techniques like gradient descent

Logistic Regression & Multiclass

- What if there c>2 classes? Can we use logistic regression?
  - Reminder: we can transform multiclass classification into a binary classification
    - One-vs-rest (OvR): Train c binary classifiers. f_i to classify class i versus not i
    - One-vs-one (OvO): Train $c(c\text{-}1)/2$ binary classifiers. $f_{i,j}$ to classify class i versus class j
- Alternative: (sometimes called softmax regression)
  - Idea: Train c classifiers each to predict a logit score z_i for each class i, then use softmax to combine into a probability distribution over the c labels
  - How? Train c distinct linear functions (each with their own weights and bias terms)
    - So the parameters are $\boldsymbol{W}$, $\boldsymbol{b}$ where $\boldsymbol{W}$ is a $c \times m$ matrix and $\boldsymbol{b}$ is a $c \times 1$ vector. $\boldsymbol{W}^{(i)}$ is the i^{th} row of the matrix containing weights for class i
  - Softmax: $f(z_j) = \frac{\exp(\frac{z_j}{T})}{\sum_{i=1}^{c} \exp(\frac{z_i}{T})}$ Here: $T > 0$ is called the *temperature*, we often set T=1
    - The softmax function is also called *normalized exponential*
  - Loss function: $L(\boldsymbol{\theta}) = -1/n \sum_i \sum_j y_i^{(j)} \log(p_i^{(j)})$ called cross-entropy loss

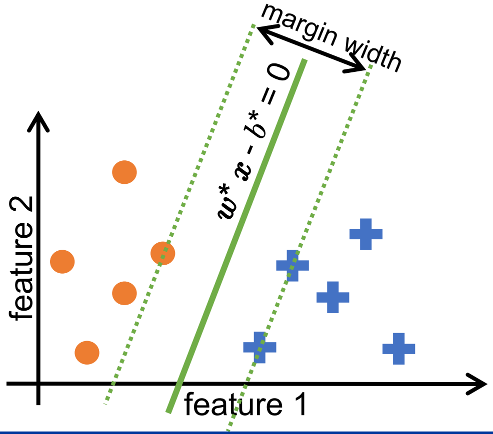
Understanding Softmax

- **Softmax**: $f(z_j) = \frac{\exp(\frac{z_j}{T})}{\sum_{i=1}^{c} \exp(\frac{z_i}{T})}$
  - $T > 0$ is called the *temperature*, we often set T=1 (e.g., for logistic regression)
    - T controls the shape of the probability distribution
    - $T \to \infty$ means uniform distribution ; $T \to 0$ means 1 for class with max logit score (0 otherwise)
- Example: suppose z =[1.78, 5.7, 4.42, -2.34, 0.9]



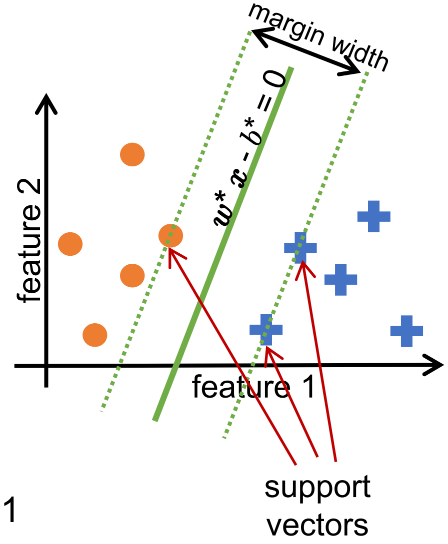
Reminder: Avocado Ripeness & SVM

- Prediction task:
  - Given the features (i.e., color, softness, texture), predict ripe (+1) or unripe/overripe (0)
- Dataset
  - Matrix $\mathbf{X}$ and the labels vector $\mathbf{y}$
- Let's use a Support Vector Machine (SVM) model:
  - We need to relabel unripe/overripe as -1, so the labels are +1 and -1
  - The SVM is represented as the hyperplane $\boldsymbol{w}\,\boldsymbol{x} - b = 0$,
    - $\boldsymbol{x}$ is a feature vector and $\boldsymbol{w}$ and b are the model's parameters
  - Define $f_{\theta}(\boldsymbol{x}) = \mathrm{sign}(\boldsymbol{w}\,\boldsymbol{x} - b)$, where $\boldsymbol{\theta} = (\boldsymbol{w}, b)$
    - If $\boldsymbol{w}\,\boldsymbol{x} - b \geq 0$, then we predict +1 (ripe)
    - Otherwise, we predict -1 (unripe/overripe)
    - Note: $\boldsymbol{w}\,\boldsymbol{x}$ is the dot-product of $\boldsymbol{w}$ and $\boldsymbol{x}$
      - $\boldsymbol{w}\,\boldsymbol{x} = w_1x_1+w_2x_2+\ldots+w_mx_m$



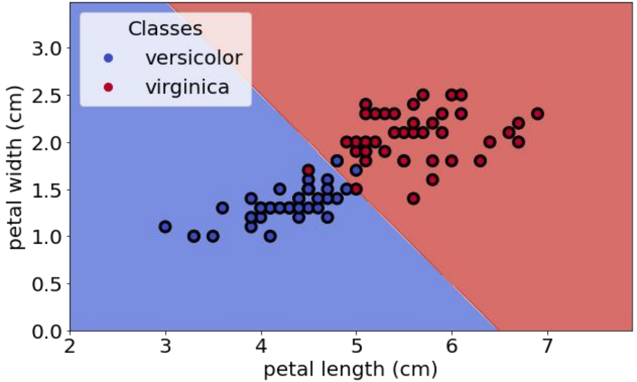
Hard-Margin SVM

- Dataset
  - Matrix $\mathbf{X}$ and the labels vector $\mathbf{y}$
  - Let $\boldsymbol{x}_i$ be the feature vector for example i and y_i be the corresponding label
- Training the SVM
  - We need to learn the optimal parameter values $\boldsymbol{w}^*$, b^* given our dataset
  - We want the hyperplane that best separates positive from negative examples
    - The one with the largest distance (called "margin") between the closest examples of each class (called support vectors)
    - The margin width is 2 / $||\boldsymbol{w}||$ so maximizing the margin means minimizing $||\boldsymbol{w}||$
  - Optimization with constraints: we want: $\boldsymbol{w}\,\boldsymbol{x_i} - b \geq +1$ if y_i=+1 and $\boldsymbol{w}\,\boldsymbol{x_i} - b \leq -1$ if y_i=-1
  - Minimize $||\boldsymbol{w}||$ subject to: $y_i(\boldsymbol{w}\,\boldsymbol{x_i} - b) \geq 1$ for i=1,2,...,n
    - Equivalent to: $\min$ 1/2 $||\boldsymbol{w}||^2$ such that $y_i(\boldsymbol{w}\,\boldsymbol{x_i} - b) \geq 1$ for i=1,2,...,n
    - Can be solved using quadratic programming optimization!
  - Note: this is the primal problem, we could instead solve the corresponding dual problem



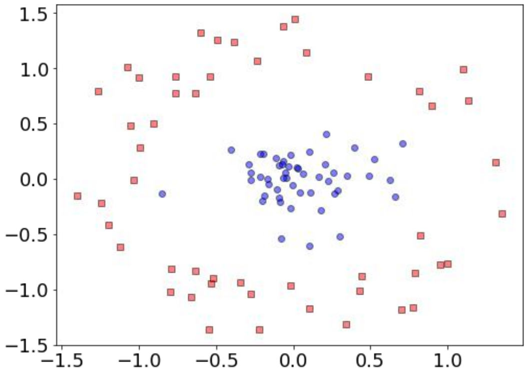
Soft-Margin SVM

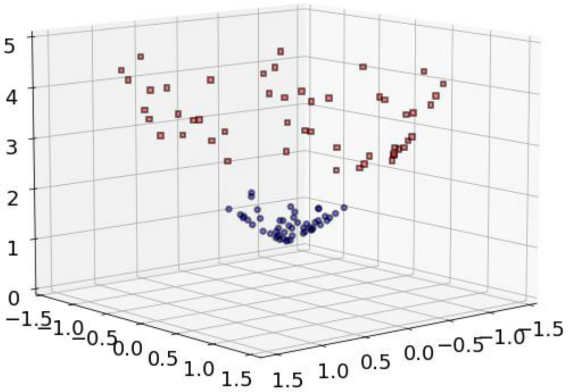
- Dataset
  - Matrix $\mathbf{X}$ and the labels vector $\mathbf{y}$
  - Let $\boldsymbol{x}_i$ be the feature vector for example i and y_i be the corresponding label
- What if the data is not linearly separable?
  - This can happen if there is noise in the data...
  - We need to relax our constraints that: $y_i(\boldsymbol{w}\ \boldsymbol{x}_i - b) \geq 1$ for i=1,2,...,n
  - Define the hinge loss: $\max(0, 1 - y_i(\boldsymbol{w}\ \boldsymbol{x}_i - b))$
    - If $\boldsymbol{x}_i$ is on the correct side of the decision boundary then the loss is 0, otherwise loss is proportional to distance from decision boundary
  - Minimize the cost function: $||\boldsymbol{w}||^2 + C \sum_i \max(0, 1 - y_i(\boldsymbol{w}\ \boldsymbol{x}_i - b))$
    - Here C is a regularization constant (hyperparameter) that controls the trade-off
    - $C \rightarrow \infty$ must separate the data! $C \rightarrow 0$ (extreme regularization) ignore data!
  - Note: the support vectors will be the points that are wrongly classified or within the margin



Non Linearity?

- What do we do if the data is inherently non linear?
  - What if add features of features?
    - For example: add ${x_1}^2$, $x_1\, x_2$, ${x_2}^3$, $\exp(x_1)$, etc
    - Note: this increases the risk of overfitting
  - Idea:
    - Transform our dataset to a higher dimensional space
    - Find a hyperplane to separate the data in this higher dimensional space!





The Kernel Trick

- Wait! It is (computationally) expensive to transform our data to higher dimensional space
  - Can we do this transformation implicitly?
  - In other words: can we only reflect the transformation only in our cost function for optimization?
    - Yes! This is called the kernel trick!
  - Suppose we have a mapping Φ such that $\Phi(\boldsymbol{x})$ is in the higher dimensional space
    - For example: if $\boldsymbol{x}=(x_1,x_2)$ we can take: $\Phi(\boldsymbol{x})=(x_1{}^2, x_1\, x_2, x_2{}^2)$
  - In the formulation of the *dual problem*, the only term involving feature vectors is their dot-product $\boldsymbol{x}_i\,\boldsymbol{x}_j$
    - So we define kernels in terms of $\boldsymbol{x}_i$, $\boldsymbol{x}_j$. That is: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i)\ \Phi(\boldsymbol{x}_j)$ (dot-product)
  - Popular kernels
    - $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i\,\boldsymbol{x}_j)^2$ ("quadratic kernel")
    - $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i\,\boldsymbol{x}_j)^k$ ("polynomial kernel" of degree exactly k)
    - $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/(2\sigma^2))$ ("RBF kernel") [feature space has infinite dimensions]
    - $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\gamma\, \boldsymbol{x}_i\, \boldsymbol{x}_j + r)$ ("sigmoid kernel")
    - Note: σ, γ, and r are hyperparameters. For RBF we can set $\gamma = 1/(2\sigma^2)$ to be consistent with Scikit-learn!

SVM & Quadratic Programming

- Hard-margin linear SVM:

Note: $\boldsymbol{w}^\top \boldsymbol{w} = ||\boldsymbol{w}||^2$
We minimize ½ $\boldsymbol{w}^\top \boldsymbol{w}$ instead of $||\boldsymbol{w}||$ because it has a nice derivative (whereas $||\boldsymbol{w}||$ is not differentiable at $\boldsymbol{w} = 0$).

$$\min_{\boldsymbol{w},b} \boxed{\tfrac{1}{2}\, \boldsymbol{w}^\top \boldsymbol{w}}$$
$$\text{subject to: } y_i(\boldsymbol{w}\, \boldsymbol{x}_i - b) \geq 1 \text{ for } i=1,2,\ldots,n$$

- Soft-margin linear SVM:

We want to minimize the overall margin violations

$$\min_{\boldsymbol{w},b,\boldsymbol{z}} \tfrac{1}{2}\, \boldsymbol{w}^\top \boldsymbol{w} + C \boxed{\textstyle\sum_i z_i}$$
$$\text{subject to: } y_i(\boldsymbol{w}\, \boldsymbol{x}_i - b) \geq 1 - z_i \text{ for } i=1,2,\ldots,n$$

Regularization hyperparameter. It defines the **tradeoff** between maximizing the margin and minimizing margin violations

$z_i \geq 0$ is the "*slack*" variable for example i. The larger z_i the more example i can violate the margin

- Both are convex and quadratic optimization problems (with linear constraints)
  - We can use quadratic programming (QP) solvers

SVM & Primal - Dual Problems

- For the dual problem to have the same solution as the primal problem
- Dual linear SVM problem:

$$\min_{\boldsymbol{\alpha}} \tfrac{1}{2} \textstyle\sum_i \sum_j \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j - \sum_i \boldsymbol{\alpha}_i$$

$$\text{subject to: } \boldsymbol{\alpha}_i \geq 0 \text{ for } i=1,2,\ldots,n \text{ and } \textstyle\sum_i \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j y_i = 0$$

- We solve this problem (e.g., using a QP solver) to find the best vector $\boldsymbol{\alpha}^*$
  - Then we transform the dual solution into the primal solution (i.e., we compute $\boldsymbol{w}^*, b^*$)

$$\boldsymbol{w}^* = \textstyle\sum_i \boldsymbol{\alpha}_i^* y_i \boldsymbol{x}_i \qquad\qquad b^* = (\boxed{n_s})^{-1} \sum_{i:\underline{\alpha(i)^* > 0}} (y_i - \boldsymbol{w}^{*T} \boldsymbol{x}_i)$$

n_s is the number of support vectors. If $\alpha_i^* > 0$ then example i is a support vector.

Kernel Trick: Why Does it Work?

- Mercer's Theorem:
  - If a function $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ satisfies *some conditions* then there exists some mapping Φ to possibly much higher dimension such that $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i)^{\mathrm{T}}\, \Phi(\boldsymbol{x}_j)$

- Why does this matter?
  - The dual formulation depends only (for the data) on the dot-product: $\boldsymbol{x}_i^T \boldsymbol{x}_j$

$$\min_{\alpha} \tfrac{1}{2} \textstyle\sum_i \sum_j \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j y_i y_j \boxed{\boldsymbol{x}_i^T \boldsymbol{x}_j} - \sum_i \boldsymbol{\alpha}_i$$

$$\text{such that: } \boldsymbol{\alpha}_i \geq 0 \text{ for } i=1,2,\ldots,n \text{ and } \textstyle\sum_i \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j y_i = 0$$

  - So we can replace that term by $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ since $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i)^{\mathrm{T}}\, \Phi(\boldsymbol{x}_j)$
  - Observe: we do not need to know how to compute Φ
    - In some cases we cannot even compute it
    - For example: RBF kernel Φ maps points to **infinite-dimensional** space

Support Vector Machines: Takeaways

- SVM is an important class of models to know about
  - It can be used to perform both linear and non-linear classification (using kernels)
  - It can be used for regression
  - It can even be used for outlier detection

- SVM is well-suited to small datasets (i.e., < 100k instances)
  - In practice it works well even if the dataset is very complex, or if it has lots of features
    - Even if the number of features far exceeds the number of instances
  - But, training can be very slow!
    - Especially if you have lots of examples or lots of features

Next Time

- Friday (2/2): Exercise

- Upcoming:
  - Homework 1 is due 2/2 by 11:59pm