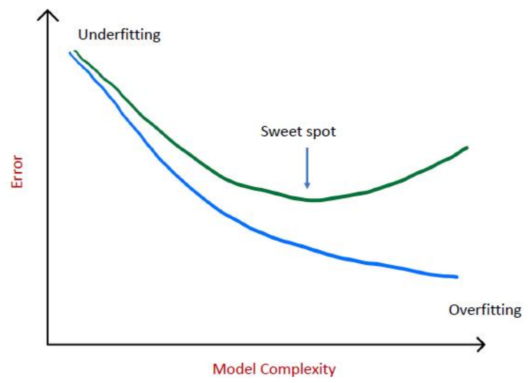
Lecture 1 Machine Learning – Lecture Notes – Ahmed Nouralla

- **Machine Learning (ML)**
    - Computer programs (models) that improve their performance at some tasks through experience (training on datasets).
    - Training data: $D = \{(x_i, y_i)\}_{i=1}^{N}$ where:
        - N: dataset size.
        - x_i: features, predictors, input to the model.
        - y_i: targets, labels, correct output for a given x_i
    - Goal of ML is to learn (estimate) how features relate to labels
        - To estimate a function $\hat{f} \sim f$ where $f(\boldsymbol{x}) = \boldsymbol{y}$ that performs well.
        - f is assumed to have some form (e.g., polynomial of a certain degree) and then we estimate parameters by choosing the ones that maximize performance.
    - Performance of a ML model is assessed using an objective function that works as the performance metric. By optimizing (minimizing/maximizing) the objective function, we make the model better.
    - **Common cost functions:** mean square error, mean absolute error
        - $\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{f}(x_i)\right)^2$, $\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\left|y_i - \hat{f}(x_i)\right|$
- **Learning algorithms**

Supervised		**Unsupervised**
Algorithm is provided with example input-output pairs to train on before testing.		Algorithm is not provided with example outputs for the given training data, instead, it has to discover patterns in the data and make assumptions (e.g., clustering tasks).
Classification problems	**Regression problems**	
y is a **qualitative** variable: can take one value from a finite set (e.g., Yes/No, Red/Green/Blue)	y is a **quantitative** variable: can take one value from an infinite set (e.g., distance, age)	

- **Overfitting vs Underfitting:**
    - **Underfitting:**
        - both errors (training and testing) are large.
        - Model complexity (flexibility) is low.
    - **Overfitting:**
        - Training error is small, test error is large.
        - Model complexity is high.
    - **Sweet spot:** the point where model complexity is adequate for the task.



- Bias-Variance tradeoff

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = Var\left(\hat{f}(x_0)\right) + \left[Bias\left(\hat{f}(x_0)\right)\right]^2 + Var(\epsilon)$$

Irreducible error, resulting from noise in the problem itself

- - **Bias:** error from erroneous assumptions in the learning algorithm (high bias → underfitting)
    - **Variance:** error from sensitivity to noise in the training set (high variance → overfitting)
    - As the model complexity increases, variance increases and bias decreases, a good model minimizes both errors.

// Revise from labs: Cross validation, one hot encoding.

Lecture 2

- **Regression Analysis**
  - Statistical techniques used in ML for estimating the relationship between a label (target) and one or more features (predictors).

- **Linear Regression (LR):**
  - Estimates $y = f(\boldsymbol{x}, \boldsymbol{w}) = \begin{bmatrix} w_1 & \cdots & w_n \end{bmatrix} \begin{bmatrix} x_1 \\ \cdots \\ x_n \end{bmatrix} + w_0$
    - $\boldsymbol{w}$ are the model parameters to be estimated.
  - **Simple LR:** $\boldsymbol{x} = [x]$
  - **Multiple LR:** $\boldsymbol{x} = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}$

Simple LR: *model is a straight line; we estimate its slope and intercept*

Multiple LR: *model is a linear combination of features; we estimate their coefficients (contribution).*

Recall from calculus: gradient of a multivariable function is the vector of its partial derivatives. $\nabla f(x, y) = (f_x, f_y)$

- **Estimating LR parameters**
  - Estimating $\boldsymbol{w}$ can be done by minimizing a cost function such as MSE (i.e., solving $\nabla_{\boldsymbol{w}}(\mathcal{L}) = \boldsymbol{0}$)
  - Example with simple LR:

$$\boldsymbol{\mathcal{L}}(\boldsymbol{w_0}, \boldsymbol{w_1}) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - (w_0 + w_1 x_i)\right)^2$$

$$\nabla_w(\mathcal{L}) = \begin{bmatrix} \dfrac{\partial \mathcal{L}}{\partial w_0} \\ \dfrac{\partial \mathcal{L}}{\partial w_1} \end{bmatrix} = \begin{bmatrix} 2w_0 + 2w_1\bar{x} - 2\bar{y} \\ 2w_1\left(\overline{x^2} - \bar{x}^2\right) + 2\bar{x}\bar{y} - 2\overline{xy} \end{bmatrix} = \boldsymbol{0} \Rightarrow \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} \bar{y} - w_1\bar{x} \\ \dfrac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \bar{x}^2} \end{bmatrix}$$

  - It can also be computed in general case with pseudoinverse as $\boldsymbol{w} = (\boldsymbol{x}^\top \boldsymbol{x})^{-\boldsymbol{1}} \boldsymbol{x}^\top \boldsymbol{y}$

- **Polynomial Regression:**
  - Estimates $y = \sum_{i=0}^{n} w_i x^i$, the model is still linear in parameters, but polynomial of the nth degree in predictors, thus it can be solved in the same way by equating the gradient to zero.
  - Multiple equations can be introduced in case of multiple features/targets.
    - $\left\{y_i = \sum_{j=0}^{n} w_j x_i^j\right\}_{i=1}^{N}$ where n: sample size, and N: feature count.

Lecture 3

- **Gradient Descent (GD)** (optimization algorithm, not a ML one)
  - Iterative method used to find the local minimum of a function, from some starting point.
  - Can be used in parameter estimation for regression, by predicting a value of w for a given training data, then iteratively reducing the cost (e.g., MSE) by moving α units in the opposite direction of the gradient vector evaluated at that point.
    - We move in the opposite direction because the gradient points to the direction of the steepest increase in the cost function.
    - **Learning rate** (α) is yet another hyper-parameter that is chosen to be a certain fraction of the gradient magnitude.

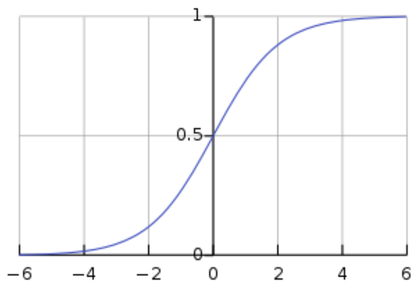
Hyper-parameters: values not learnt from data, but chosen by the ML engineer, they affect the model performance, we tune them by testing, computing error, then modifying parameters and testing again.

  - **Example with two parameters:**
    - Cost function: $\boldsymbol{\mathcal{L}(w_0, w_1)} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - (w_0 + w_1 x_i)\right)^2$
    - **Gradient descent:**
      - Compute value of $\mathcal{L}(w_0, w_1)$ for initial (can be random) w_0, w_1
      - Update $\begin{cases} w_0 = w_0 - \alpha\frac{\partial\mathcal{L}}{\partial w_0} = w_0 - \frac{\alpha}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i) \\ w_1 = w_1 - \alpha\frac{\partial\mathcal{L}}{\partial w_1} = w_1 - \frac{\alpha}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)\, x_i \end{cases}$ and recompute $\mathcal{L}$.
      - Repeat until minimum value of $\mathcal{L}$ is reached.
  - For multiple predictors, same algorithm can be applied.

- **Logistic function**: an S-shaped (sigmoid) curve following the equation

$$\sigma(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

  - Where:
    - x_0: value of the midpoint (used for shifting)
    - L: the curve's maximum value (or minimum if negative, the maximum in that case is 0).
    - k: logistic growth rate (steepness of the curve)

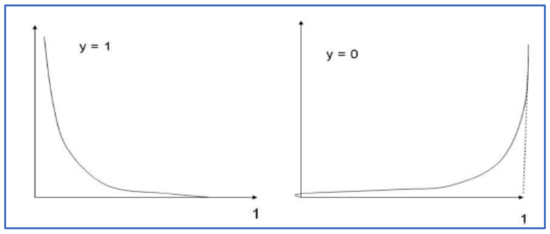


Standard logistic sigmoid function where $L = 1, k = 1, x_0 = 0$

  - Nice property of the **standard** logistic function:
    - $\frac{d}{dx}\sigma(x) = \sigma(x)\left(1 - \sigma(x)\right)$

- **Logistic Regression (logit model)**
  - Assumes f to be a logistic (instead of polynomial) function.
  - It helps solving binary (can be extended) classification problems since the standard logistic function can be used to calculate probability (bounded between 0 and 1) that a given input feature belongs to a certain class.
  - The underlying technique is still regression (hence the name), we still estimate parameters by minimizing some cost function.
  - $\sigma(x_0)$ gives the threshold such that the function is mirrored around this point.

- **Log (logistic) loss function**
  - Used as the evaluation metric for logistic regression
  - We use it since it has the curve we need (small error for correct prediction, higher penalty as we go away from it.
    - **Let:**
      - $y \in \{0, 1\}$ be the true label for a given feature x.
      - $p(x) = P(y = 1 \mid x) = \frac{1}{1+e^{-z(x)}} = \sigma(z(x))$ be the probability distribution (model) proposed by the algorithm ($z(x)$ can be linear or polynomial in features, but strictly linear in parameters).
    - **Then** the log-loss function is defined as:
      $$\mathcal{L}(y, p) = -[y \ln(p) + (1-y)\ln(1-p)]$$
    - **Note:** we can't use MSE anymore since the sigmoid function is non-linear (hence, the loss curve is not necessarily convex)
  - We can again use GD to minimize such function and find the best parameters.



Log-loss function with p(x) on the x-axis and $\mathcal{L}$ on the y-axis for cases when y=1 or y=0

- **Example with a single predictor**
  - Predict whether a patient has a certain disease ($y = 1$) given his/her age x
  - $P(y = 1 \mid x) = \frac{1}{1+e^{-z(x)}} = \frac{1}{1+e^{-(w_0+w_1x)}}$
  - $\mathcal{L} = -\left[y \ln \sigma(z(x)) + (1-y)\ln\left(1-\sigma(z(x))\right)\right]$
    - For multiple features and labels, multiple equations can be introduced.
  - Now we need to find values for w_0, w_1 that minimize the cost function.
  - Let's do it for a general $z = \sum_{i=0}^{n} w_i x_i$ with $x_0 = 1$

$$\nabla_w(\mathcal{L}) = \left\{\frac{\partial \mathcal{L}}{\partial w_i}\right\}_{i=0}^{n} = -\left\{\frac{y}{\sigma(z)} * \frac{\partial(\sigma(z))}{\partial z} * \frac{\partial z}{\partial w_i} + \frac{1-y}{1-\sigma(z)} * \frac{-\partial(\sigma(z))}{\partial z} * \frac{\partial z}{\partial w_i}\right\}_{i=0}^{n}$$

$$= -\left\{\frac{\partial z}{\partial w_i}\left(y * (1-\sigma(z)) - (1-y) * \sigma(z)\right)\right\}_{i=0}^{n} = \left\{-x_i\left(y - \sigma(z)\right)\right\}_{i=0}^{n}$$

  - For the case with 2 parameters w_0, w_1 we have $\nabla_w(\mathcal{L}) = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial w_0} \\ \frac{\partial \mathcal{L}}{\partial w_1} \end{bmatrix} = \begin{bmatrix} \frac{1}{1+e^{-(w_0+w_1x)}} - y \\ \left(\frac{1}{1+e^{-(w_0+w_1x)}} - y\right) x_1 \end{bmatrix}$
    - Again, we don't equate the gradient to 0 and solve, we want the global min, not all local/global max/min points.
  - Instead, we use GD with several starting points, we keep updating the parameters following the rule $w_i = w_i - \alpha \mathcal{L}_{w_i}$ until we reach the minimum.

- **Types of GD:**

Batch GD	**Stochastic GD**	**Mini-Batch GD**
Uses entire dataset (tries each point as a starting point) to guarantee that we reach global min.	Uses one random example and pray that we reach global min from it. If we reached a local min, it's still ok.	Uses a fixed-size subset of the dataset that is chosen smartly.
Low performance	High performance.	Balanced.

- **Evaluating classifier models:**
  - We define a certain probability threshold ($0 < \text{Th} < 1$) and say that $\hat{y} = \begin{cases} 1 & p(x) > \text{Th} \\ 0 & \text{otherwise} \end{cases}$
  - Then we evaluate our model based on its correctness using different metrics.
  - **Confusion matrix:** helps evaluating the model by arranging results of the training set in a table.
    - **Example:** FP = number of cases for which the model guessed that $y = 1$ but in fact $y = 0$

Actual \ Predicted	**Negative**	**Positive**
Negative	True Negative (TN)	False Positive (FP)
Positive	False Negative (FN)	True Positive (TP)

Commonly used metrics (*)			
$\text{Accuracy} = \frac{\text{\# correct guesses from test set}}{\text{test set size}}$	$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$	$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$	$\text{F1\_Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$
- Accuracy is not a good metric for imbalanced test set (maybe the model always does majority guessing). - Using only precision or recall might as well be in favor of a majority guessing model (thus we use F1)			

Lecture 4

- **Classifier:** a function C that takes data (value of one or more predictors) and returns a class $C(x) = y = c_k$

- **Bayes Equation:** $P(A \mid B) = \frac{P(AB)}{P(B)}$
    - **In general:** $P(AB) = P(B \mid A)P(B)$
    - **For independent events:** $P(AB) = P(A)P(B)$ and the equation becomes $P(A \mid B) = P(A)$

Law of total probability

Let $A_1, A_2, \dots A_n$ be a set of

1. Pairwise exclusive (disjoint) events = no two events happen together = $P(A_i A_j) = 0, i \neq j$.
2. Collectively exhaustive = at least one of them occurred = $\bigcup_{i=1}^{n} P(A_i) = 1$

Then, for any event B in the same probability space we have:

$$P(B) = P(B|A_1)P(A_1) + \dots + P(B|A_n)P(A_n)$$

In our case: $P(A_i) = P(y = c_i)$

- **Bayes Classifier**:
    - A classifier that applies Bayes equation to classify data to the most probable class.

$$C(x) = \underset{c_k}{\operatorname{argmax}} P(y = c_k \mid \boldsymbol{X} = \boldsymbol{x})$$

    - For a given feature value(s), we find the class c_k that maximizes the probability of x being classified as c_k
    - To find the conditional probability, we use Bayes equation and the law of total probability as follows:

$$P(y = c_k \mid \boldsymbol{X} = \boldsymbol{x}) = \frac{P(y = c_k, \boldsymbol{X} = \boldsymbol{x})}{P(\boldsymbol{X} = \boldsymbol{x})} = \frac{P(\boldsymbol{X} = \boldsymbol{x} \mid y = c_k)P(y = c_k)}{\sum_{i=1}^{n} P(\boldsymbol{X} = \boldsymbol{x} \mid y = c_i)p(y = c_i)}$$

    - **Naïve Bayes:** assuming that features are independent from each other equation becomes:

$$P(y = c_k \mid \boldsymbol{X} = \boldsymbol{x}) = \frac{P(\boldsymbol{X} = \boldsymbol{x} \mid y = c_k)P(y = c_k)}{\sum_{i=1}^{n} P(\boldsymbol{X} = \boldsymbol{x} \mid y = c_i)p(y = c_i)} = \frac{P(y = c_k)\prod_{i=1}^{n} P(X_i = x_i \mid y = c_k)}{\sum_{i=1}^{n} P(\boldsymbol{X} = \boldsymbol{x} \mid y = c_i)p(y = c_i)}$$

    - The denominator doesn't depend on the c_k we choose (it's summing all possibilities anyway), so the classifier needs only to maximize the argument for the numerator.

- **Example with two predictors:**
    - Given a dataset of email senders (X_1) and email subjects (X_2) and whether email was a spam or not (y), use Bayes classifier to predict $y_{new} \in \{\text{spam, not\_spam}\}$ for a given $x = (\text{A, B})$
    - To apply Naïve Bayes classifier, we need to assume that email sender has nothing to do with email subject, we also need to infer some values for calculations:
        - $P(y = \text{spam})$ can be inferred by dividing the number of samples in the training data where $y = \text{spam}$ over the total training set size (i.e., number of spams / total training emails)
        - $P(X_1 = A \mid y = \text{spam}), P(X_2 = B \mid y = \text{spam})$ can be inferred from frequencies in the data set.
            - Number of spam emails in the training set from sender A | with subject B divided by the total number of spam emails.

    - *Do the math*: $C((A, B)) = \underset{c_k \in \{\text{spam, not\_spam}\}}{\operatorname{argmax}} P(y = c_k \mid \boldsymbol{X} = (A, B))$
        - That is, calculate the probabilities of the email being spam and "not spam" given the sender and subject, then output the label that yielded the maximum.

- **Naïve Bayes Classifier**
    - **Advantages:**
        - Easy and fast to use for binary/multi-class classification tasks
        - Typically performs better that logistic regression and needs less training data.
        - Allows online learning (new samples can be utilized without the need to retrain the model)
    - **Disadvantages:**
        - If a feature was not encountered in the training set, the model will assign 0 probability to it.
            - Use a smoothing technique (e.g., Laplace estimation)
        - Multiplication of many small numbers (probabilities) can result in underflow ($0.0001 \cong 0$)
            - We can take the argmax on log(P) instead of P and use $\log(xy) = \log(x) + \log(y)$

- **K-Nearest Neighbors (KNN)**
    - **Other names:** instance-based/lazy/non-parametric learning.
    - Statistical technique used by ML algorithms for classification problems.
    - For a given x_{new} to be labeled, we find the closest k points x_i to it from the training set.
    - We assign to x_{new} the most common label among the k points observed.
    - With $K = 1$, classifier is $C(x_{new}) = y\left(\underset{x_i \in X}{\operatorname{argmin}} \operatorname{dist}(x_i, x_{new})\right)$
        - Distance can be the Euclidean distance between vectors, or any other metric defined by the ML engineer.
    - No training needed, but computational cost for testing can be high ($O(nd + n\log n)$ for the naïve implementation that loops over the whole training set with d predictors (dimensions)
    - K is yet another hyper-parameter
        - Small k is good at capturing complicated patterns, but may overfit
        - Large k may underfit.
        - Optimal choice is chosen using a validation set, and should be less that $\sqrt{n}$

- **K-fold cross-validation:**
    - Splits the training set into K-folds (sets).
    - Iterate over the dataset K times, each time we use one-fold for validation and the rest for training.
    - We may take the average error as an overall estimation.

- **Regularization**
    - Measuring the success of ML model based on how good it fits the training data (MSE, RSS, or any other cost function), but making sure we avoid overfitting the data (high variance) by introducing a penalty (bias) to the model (using a non-negative factor λ).
    - λ is yet another hyper-parameter (can be tuned by cross validation), a good choice helps the model avoid overfitting, it can happen that $\lambda = 0$ (not using regularization at all) is ok.
    - **Ridge regression:** a regression model that uses L2 regularization:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(y_i - w^\top \cdot x_i)^2 + \lambda \sum_{j=1}^{p} w_j^2$$

    - **Lasso regression:** a regression model that uses L1 regularization:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(y_i - w^\top \cdot x_i)^2 + \lambda \sum_{j=1}^{p} |w_j|$$

Unlike L_2, L_1 produces sparse models (performs variable selection) since it can set coefficients of non-important features to 0.

Lecture 5

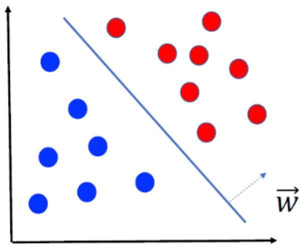
- **Constrained Optimization Problems**
  - Maximization/Minimization problems where the function $f(x)$ is subject to some constraint $g(x)$
  - Solved using Lagrange multipliers:
    - To optimize $f(\boldsymbol{x})$ subject to $g(\boldsymbol{x}) = 0$ we can instead solve for $\begin{cases} \nabla_{\mathbf{x}}\, [f - \lambda\, g(\boldsymbol{x})] = \mathbf{0} \\ g(\boldsymbol{x}) = 0 \end{cases}$

- **Principal Component Analysis (PCA)**
  - Helps visualizing high dimensional data by projecting it to a lower dimension while retaining original structure as much as possible.
  - The data points (samples) should be shifted to have 0 mean.
  - PCA works by choosing a set of vectors w_I that maximize variance in project (lower dimension) space. These vectors should be **orthogonal** and have **unit** length.
  - **Let:**
    - $X = \{(x_i) \mid x_i \in \mathbb{R}^p\}_{i=1}^{n}$ be the dataset of p features that we want to project into $d < p$ dimensions.
    - $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$ be the mean value of the sample.
    - $C = \frac{1}{n}\sum_{i=1}^{n} x_i\, x_i^{\top}$ be the sample covariance matrix
  - **Then** PCA does the following steps:
    - Transform the data to have zero mean by subtracting μ from each point.
    - Find (eigenvector w, eigenvalue λ) pairs of C
      - Recall: $|C - \lambda I| = 0, (C - \lambda I)w = 0$
      - We know that $Cw = \lambda w = \sigma^2 w$
    - Find the eigenvectors $W = w_1, w_2, \dots, w_n$ corresponding to d highest eigenvalues.
    - Compute $X' = XW$

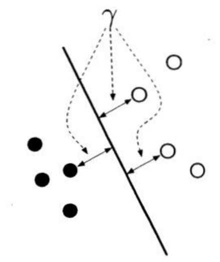
Lecture 6

- **Hyperplanes in $\mathbb{R}^n$**
  - Can be described by the following equation $w_0 + w_1 x_1 + \cdots + w_n x_n = 0$
    - If $w_0 = 0$, then the hyperplane passes through the origin.
    - The vector $\boldsymbol{w} = (w_1, \dots, w_n)$ is normal (orthogonal) to the hyperplane.
  - Can be determined by a set of points $\{x_i\}_{i=1}^{n}$
  - $n = 1$ (point), $n = 2$ (line), $n = 3$ (plane), $n > 3$ (hyperplane)

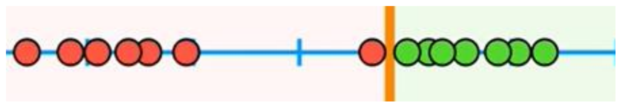
- **Support Vector Classifier (SVC)**



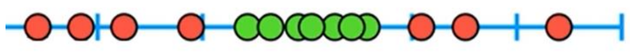
  - Used to solve binary classification problems by finding the separating hyperplane (divides data points into two groups) that maximizes margin.
    - **Margin:** perpendicular distance from decision boundary to the closest point on either of its sides.
  - **Decision function:** $y_{new} = \text{sgn}(w_0 + w_1 x_{new,1} + \cdots, w_n x_{new,n})$
    - **Interpretation:** by plugging a test point x_{new} into the equation for a separating hyperplane and checking the sign of the result we know whether the point lies on the hyperplane or one of its sides (positive or negative side) and classify the point based on that.



  - **The hyperplane that maximizing margin is the one that minimizes $||w||$ subject to the constraint $y_i * (w_0 + \boldsymbol{w}^\top \boldsymbol{x_i}) \geq 1 \;\; \forall i \in [1, n]$ (Proof for 2D case → check the slides).**
    - **Interpretation**: we choose the normal vector with minimal magnitude such that the hyperplane it defines correctly classifies **all** data points in the training set.
    - $y_i \in \{-1, 1\}$ is the correct label for the set of feature values $\boldsymbol{x_i}$
  - The problems $\underset{w}{\text{argmax}} \frac{1}{||w||}$ and $\underset{w}{\text{argmin}} \frac{1}{2}||w||^2$ have strong duality (solution to the first is the solution to the second), and the second one is easier to solve (using Lagrange multipliers) so we solve it instead.
- **Issues with SVC**

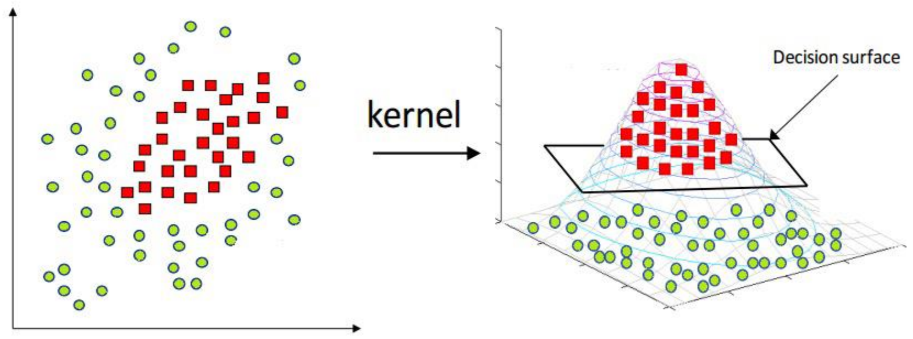


  - **Sensitivity to noise (outliers)**
    - **Solution:** Soft-Margin SVC.
  - **Non-linear data**



    - **Solution:** Support Vector Machines (SVMs) that use Kernel functions.
- **Soft-Margin SVC**
  - Reduces variance (avoid overfitting) by intentionally increasing bias (misclassifications).
    - That is, we allow the model to make a certain number of mistakes, defined by slack variables (ξ_i)
    - Now the constraint becomes $y_i * (w_0 + \boldsymbol{w}^\top \boldsymbol{x}) \geq 1 - \xi_i \;\; \forall i \in [1, n], \xi_i \geq 0$
      - $\xi_i = 0$ → sample is correctly classified (lies in the correct margin, closer to its class)
      - $\xi_i > 0$ → sample is correctly classified (wrong margin, closer to the other class)
      - $\xi_i > 1$ → sample is wrongly classified (wrong side of the hyperplane)
    - And the constrained optimization problem becomes $\underset{w}{\text{argmin}} \frac{1}{2}||w||^2 + C \sum_{i=1}^{n} \xi_i$ where C is yet another hyper-parameter to be chosen by cross-validation.

- **Support Vector Machines (SVM)**
    - Projects the data into a higher dimensional space by introducing more features (axes).
    - The new features result from the application of kernel function to the original features.
    - **Commonly used kernel functions**
        - Linear Kernel.
        - Polynomial Kernel.
        - RBF Kernel.
        - Sigmoid function.



Lecture 7

- **Feature Learning (in contrast with feature engineering)**
  - Techniques that allow ML model to automatically discover the representations needed for feature detection or classification from raw data.
  - **Hierarchal features** mean building features from another features
    - **E.g.,** pixels construct segments, segments construct shapes, shaped construct numbers.
  - We typically want to construct high-level features as a non-linear combination of lower-level ones.
- **Artificial Neural Networks (ANN) – Playlist 1, Playlist 2**
  - From graph terminology to ANN terminology
    - Layered network of nodes (neurons) with the first one being the input layer (unprocessed input), several hidden layers (black box), finally, an output layer (results)
    - A neuron contains just a number (activation)
    - The layers are connected by weighted edges (synapses) representing the operations being done on the data.
- **Feedforward Neural Networks**
  - Information only moves from input layer to output layer (in one direction)
  - Deep NNs is simply a feedforward NN with many layers.
  - Value (activation) associated with each neuron is given by

$$a_i^l = \sigma\left(\sum_k w_{ik}^l a_k^{l-1} + b_i^l\right) = \sigma(w^l a^{l-1} + b^l) = \sigma(z^l)$$

    - **Interpretation:** Activation of neuron i in layer l = **activation function** (weighted sum of activations from all neurons in the previous layer + bias of neuron i in layer l)
    - w_{ij}^l is the weight associated with the edge from neuron j in layer $l-1$ to neuron i in layer l.
    - Activation function maps its input to $[0,1]$, examples:
      - Sigmoid function
      - Rectified Linear Unit (ReLU), Leaky ReLU, Exponential ReLU.
    - **Neural Learning:** finding optimal weights and biases through gradient descent.
- **Backpropagation (BP)**
  - Algorithm for training feedforward neural networks.
  - Computes $\nabla_w C$ that is necessary for the gradient descent to work.
  - Computation uses the chain rule: $\frac{\partial C}{\partial w^l} = \frac{\partial z^l}{\partial w^l} * \frac{\partial a^l}{\partial z^l} * \frac{\partial C}{\partial a^l} = a^{l-1} * \sigma'(z^l) * 2(a^l - y)$
  - **Algorithm**:
    - **For each training example x:**
      - **Feedforward:** for l in $\{2, \dots, L\}$ calculate $a^l = \sigma(z^l) = \sigma(w^l a^{l-1} + b^l)$
        - a^1: data (activation) at input layer, a^L: activation at output layer.
      - **Calculate output error:** compute $\delta^L = \nabla_a C_x \cdot \sigma'(z^l)$
      - **Backpropagate error:** for l in $\{L-1, \dots, 2\}$ calculate $\delta^l = (w^{l+1})^\top \delta^{l+1} \cdot \sigma'(z^l)$
    - **Update weights and biases using GD:**
      - for l in $\{L, \dots, 2\}$ update $w^l = w^l - \alpha \sum_x \delta^l (\alpha^{l-1})^\top$, $b^l = b^l - \alpha \sum_x \delta^l$