Machine Learning: Models, Training, and Validation

Overview

- Tuesday
  - Machine learning algorithms:
    - Decision Tree
    - kNN
    - Linear/logistic Regression
    - SVM
  - Ensemble learning: bagging, boosting
  - Model selection and regularization
    - Feature selection, shrinkage, dimension reduction
- Thursday
  - Model validation: cross-validation, metrics
  - Practical issues: Hyperparameter tuning, feature scaling, data augmentation, imbalanced dataset, high feature dimensions

Confusing Terms: $Y = b_0 + b_1X_1 + b_2X_2$

- Independent variables: X_1, X_2
  - Depending on the context, an independent variable is sometimes called a "predictor variable", regressor, covariate, "controlled variable", "manipulated variable", "explanatory variable", exposure variable (see reliability theory), "risk factor" (see medical statistics), **"feature" (in machine learning and pattern recognition)** or "input variable"

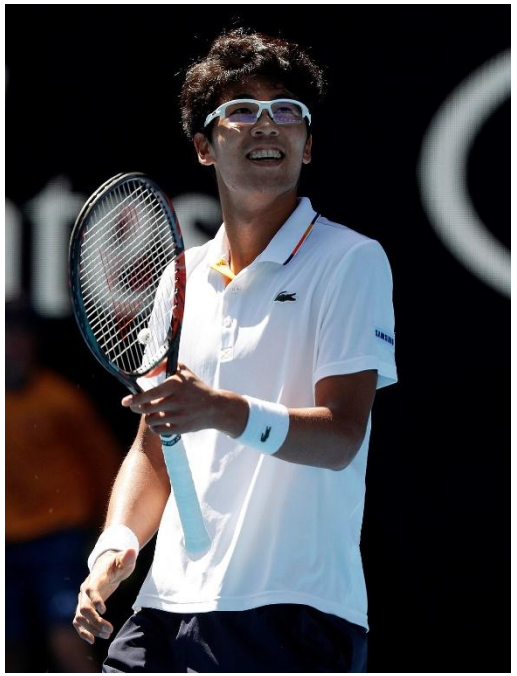
- Dependent variable: Y
  - Depending on the context, a dependent variable is sometimes called a "response variable", "regressand", "criterion", "predicted variable", "measured variable", "explained variable", "experimental variable", "responding variable", "outcome variable", "output variable", "target" or "label"

https://en.wikipedia.org/wiki/Dependent_and_independent_variables

Overview

- Tuesday
  - Machine learning algorithms:
    - Decision Tree
    - kNN
    - Linear/logistic Regression
    - SVM
  - Ensemble learning: bagging, boosting
  - Model selection and regularization
    - Feature selection, shrinkage, dimension reduction
- Thursday
  - Model validation: cross-validation, metrics
  - Practical issues: Hyperparameter tuning, feature scaling, data augmentation, imbalanced dataset, high feature dimensions

Example: Weather & Tennis Play



Example: Weather & Tennis Play

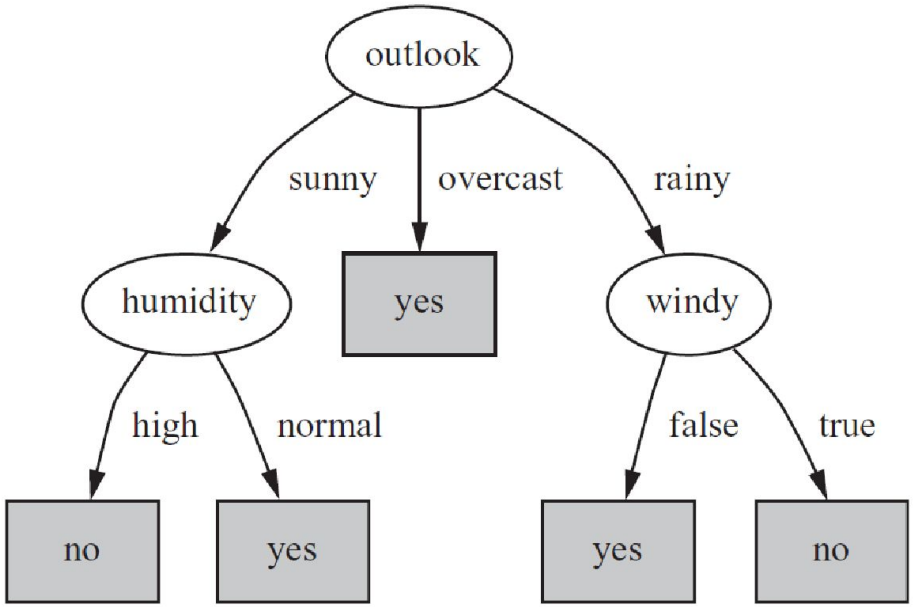
Outlook	Temperature	Humidity	Windy	Play
Sunny	hot	high	false	no
Sunny	hot	high	true	no
Overcast	hot	high	false	yes
Rainy	mild	high	false	yes
Rainy	cool	normal	false	yes
Rainy	cool	normal	true	no
Overcast	cool	normal	true	yes
Sunny	mild	high	false	no
Sunny	cool	normal	false	yes
Rainy	mild	normal	false	yes
Sunny	mild	normal	true	yes
Overcast	mild	high	true	yes
Overcast	hot	normal	false	yes
Rainy	mild	high	true	no

Q: Today we have (sunny, hot, high, true). Shall I play tennis today?

Examples from "Data Mining: Practical Machine Learning Tools and Techniques"

Decision Tree

Q: Today we have (sunny, hot, high, true). Shall I play tennis today?



Features: Outlook, Temperature, Humidity, Windy
(Temperature feature is irrelevant)

Examples from "Data Mining: Practical Machine Learning Tools and Techniques"

Decision Tree

- C4.5 is the most widely used decision tree classifier
  - J48, an open-source Java implementation of C4.5 (Weka)
  - CART (Classification and Regression Trees) (scikit-learn)

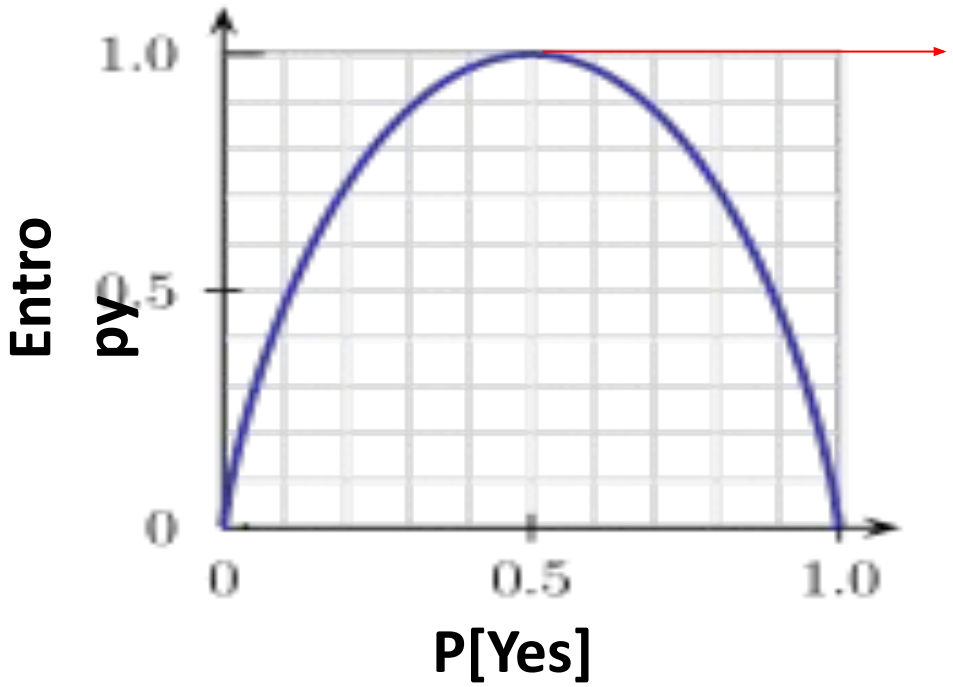
- Automatically building a tree based on the concept of information gain
  - Information gain of an attribute: the information value of creating a branch on that attribute

- Weather example: which attribute to split? (e.g., Outlook, Temperature, Humidity, Wind)

Examples from "Data Mining: Practical Machine Learning Tools and Techniques"

Decision Tree

- Info value = Info([#yes, #no]) = entropy(P[yes], P[no])
- Entropy(P[yes], P[no]) = H(P[yes], P[no])
  – P[yes]* log_2 P[yes] – P[no]*log_2 P[no]
  (= expected uncertainty of a random variable)



$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log_b \mathrm{P}(x_i)$$

$$= -\sum_{i=1}^{2} \frac{1}{2} \log_2 \frac{1}{2}$$

$$= -\sum_{i=1}^{2} \frac{1}{2} \cdot (-1) = 1$$

Decision Tree

- Info value = Info([#yes, #no]) = entropy(P[yes], P[no])
- Entropy(P[yes], P[no]) = -P[yes]* log_2 P[yes] – P[no]*log_2 P[no] (= expected uncertainty of a random variable)

Outlook	Temperature	Humidity	Windy	Play
Sunny	hot	high	false	no
Sunny	hot	high	true	no
Overcast	hot	high	false	yes
Rainy	mild	high	false	yes
Rainy	cool	normal	false	yes
Rainy	cool	normal	true	no
Overcast	cool	normal	true	yes
Sunny	mild	high	false	no
Sunny	cool	normal	false	yes
Rainy	mild	normal	false	yes
Sunny	mild	normal	true	yes
Overcast	mild	high	true	yes
Overcast	hot	normal	false	yes
Rainy	mild	high	true	no

P[yes] = 9/14, P[no] = 5/14

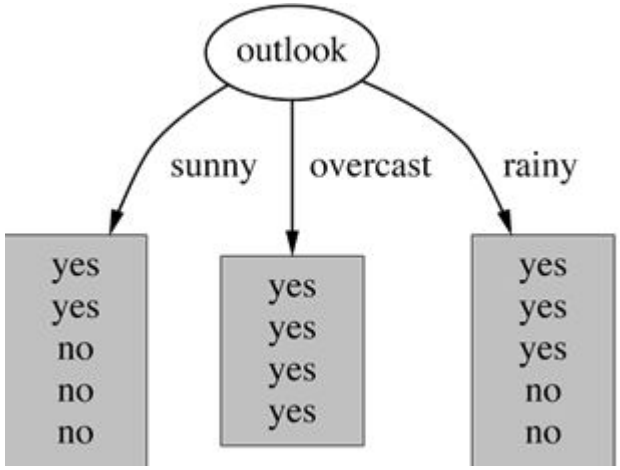
Root: Info([9,5]) = $-9/14*\log_2 9/14 -5/14*\log_2 5/14$ = 0.940 bits

Decision Tree

- Information gain of an attribute: the information value of creating a branch on that attribute
  - Entropy of a random variable: how many bits to represent the random variable? (measure of randomness)
  - Info Gain = Entropy of the class – entropy of the class (given an attribute)

Decision Tree

- How much **information gain** can we get by knowing **outlook**?



Info([2,3]) = 0.971 bits Info([4,0]) = 0 bit Info([3,2]) = 0.971 bits

Outlook	Temperature	Humidity	Windy	Play
Sunny	85	85	false	no
Sunny	80	90	true	no
Overcast	83	86	false	yes
Rainy	70	96	false	yes
Rainy	68	80	false	yes
Rainy	65	70	true	no
Overcast	64	65	true	yes
Sunny	72	95	false	no
Sunny	69	70	false	yes
Rainy	75	80	false	yes
Sunny	75	70	true	yes
Overcast	72	90	true	yes
Overcast	81	75	false	yes
Rainy	71	91	true	no

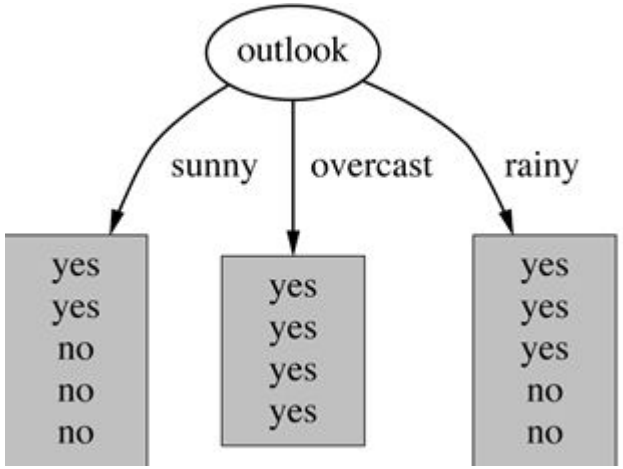
Root: Info([9,5]) = 0.940 bits
(cf: p[yes] = 9/14, p[no] = 5/14)

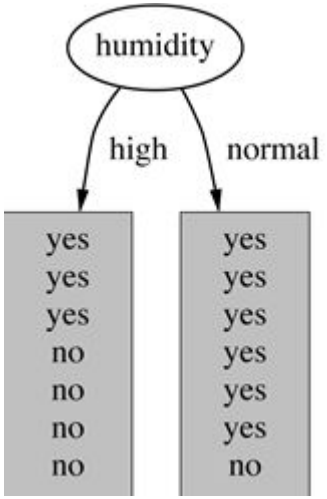
Avg info = 5/14*Info([2,3]) + 4/14*Info([4,0])
+ 5/14*Info([3,2]) = 0.693 bits

Information gain = 0.940 – 0.693 = 0.247

Decision Tree

- Which variable to use: Outlook vs. humidity?





Info([2,3]) = 0.971 bits Info([4,0]) = 0 bits Info([3,2]) = 0.971 bits Info([3,4]) = 0.98 Info([6,1]) = 0.59

Average information = 5/14*Info([2,3]) + 4/14*Info([4,0]) + 5/14*Info([3,2]) = 0.693

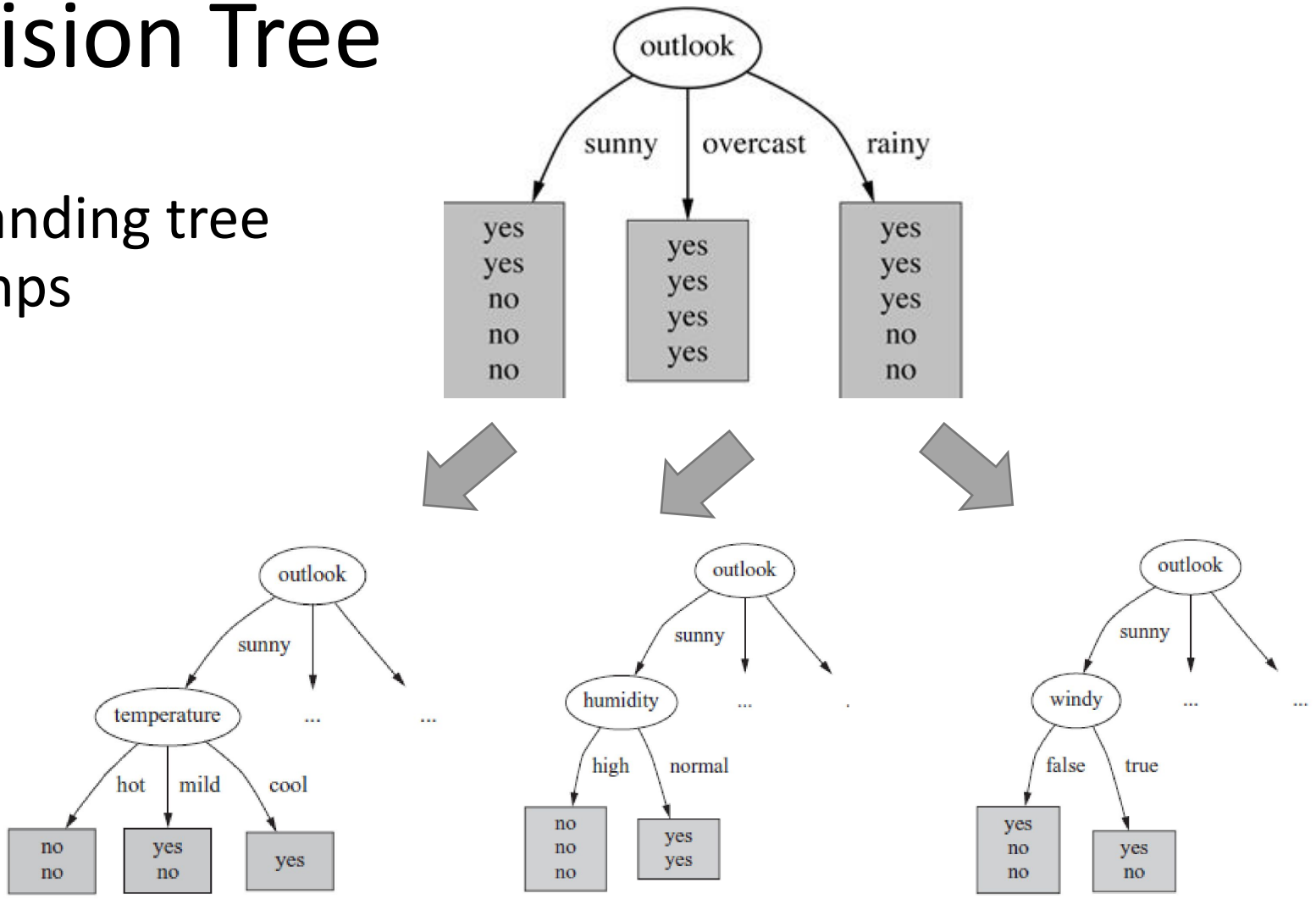
Information gain = 0.940 – 0.693 = 0.247

Average information = 7/14*Info([3,4]) +7/14*Info([6,1]) = 0.785

Information gain = 0.940 – 0.785 = 0.152

Decision Tree

- Expanding tree stumps



Overview

- Machine learning algorithms:
  - Decision Tree
  - kNN
  - Linear/logistic Regression
  - SVM
- Ensemble learning: bagging, boosting
- Model selection and regularization
  - Feature selection, shrinkage, dimension reduction
- Model validation: cross-validation, metrics
- Practical issues: Hyperparameter tuning, feature scaling, data augmentation, imbalanced dataset, high feature dimensions

Nearest Neighborhood

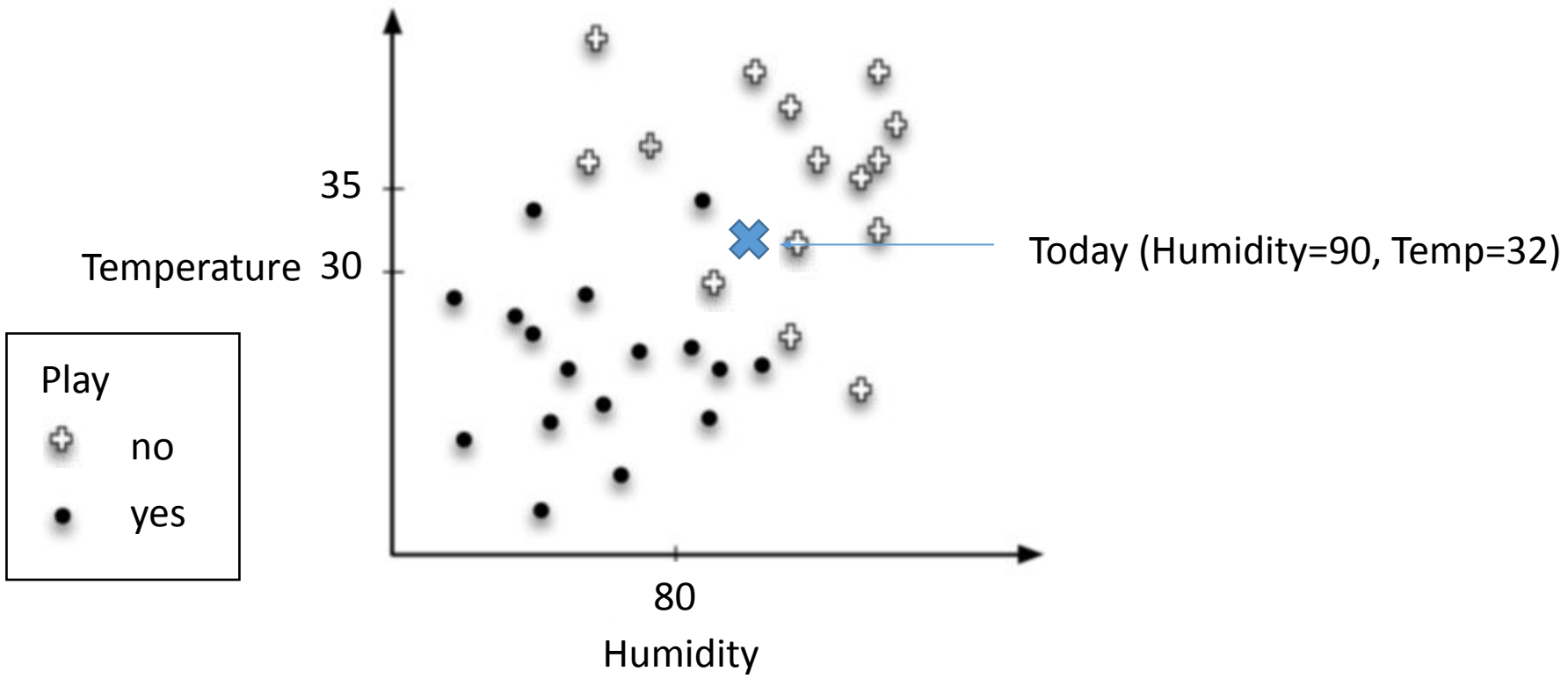
- Training examples are stored in the database
- For a given instance, find the closest member of the training set; and classify the instance as the class of the found member
- k-nearest neighborhoods (k-NN): use *k* nearest neighborhoods to make a decision (i.e., majority voting)
- That's why it is called "instance-based learning" (sometimes called "memory-based learning")

Nearest Neighborhood

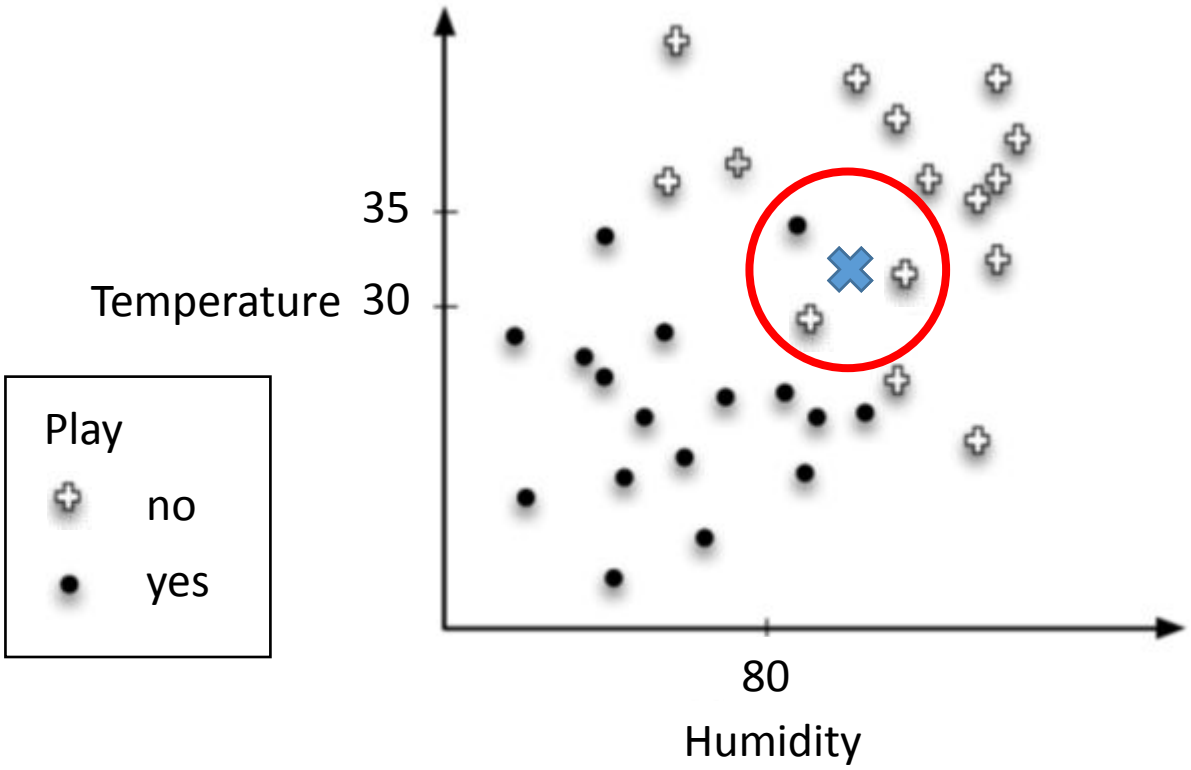
- Distance function is required: Euclidian, Jaccad similarity (categorical data), etc.

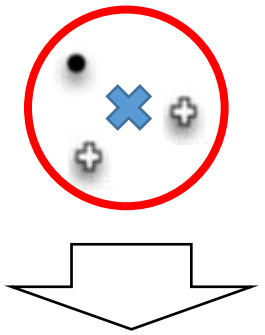
- **Normalize** all the attributes to avoid potential bias

Nearest Neighborhood



Nearest Neighborhood





Majority voting
P(Play=Yes) = 1/3
P(Play=no) = 2/3

Answer = No

Nearest Neighborhood: Summary

- Instance-based learning method:
  - find the closest member of the training set
  - classify the instance as the class of the found member
- *k*-nearest neighborhoods (k-NN): use *k* nearest neighborhoods to make a decision (i.e., majority voting)
- Distance measure: Euclidian, Jaccad similarity (categorical data), etc.
- Normalize all the attributes to avoid potential bias

Overview

- Machine learning algorithms:
  - Decision Tree
  - kNN
  - Linear /logistic Regression
  - SVM
- Ensemble learning: bagging, boosting
- Model selection and regularization
  - Feature selection, shrinkage, dimension reduction
- Model validation: cross-validation, metrics
- Practical issues: Hyperparameter tuning, feature scaling, data augmentation, imbalanced dataset, high feature dimensions

Linear Regression

- Simple linear approach for predicting a quantitative response *Y* on the basis of a single predictor variable X □ $\mathbf{Y \approx \beta_0 + \beta_1 X}$
- *Example:*
  - X may represent “TV advertising” amount and Y may represent sales.
  - Then we can regress sales onto TV by fitting the model **sales ≈ β0 + β1 × TV**
- Linear regression aims to find estimators β_0 and β_1 from the dataset

from **An Introduction to Statistical Learning, Chapter 3**

Linear Regression

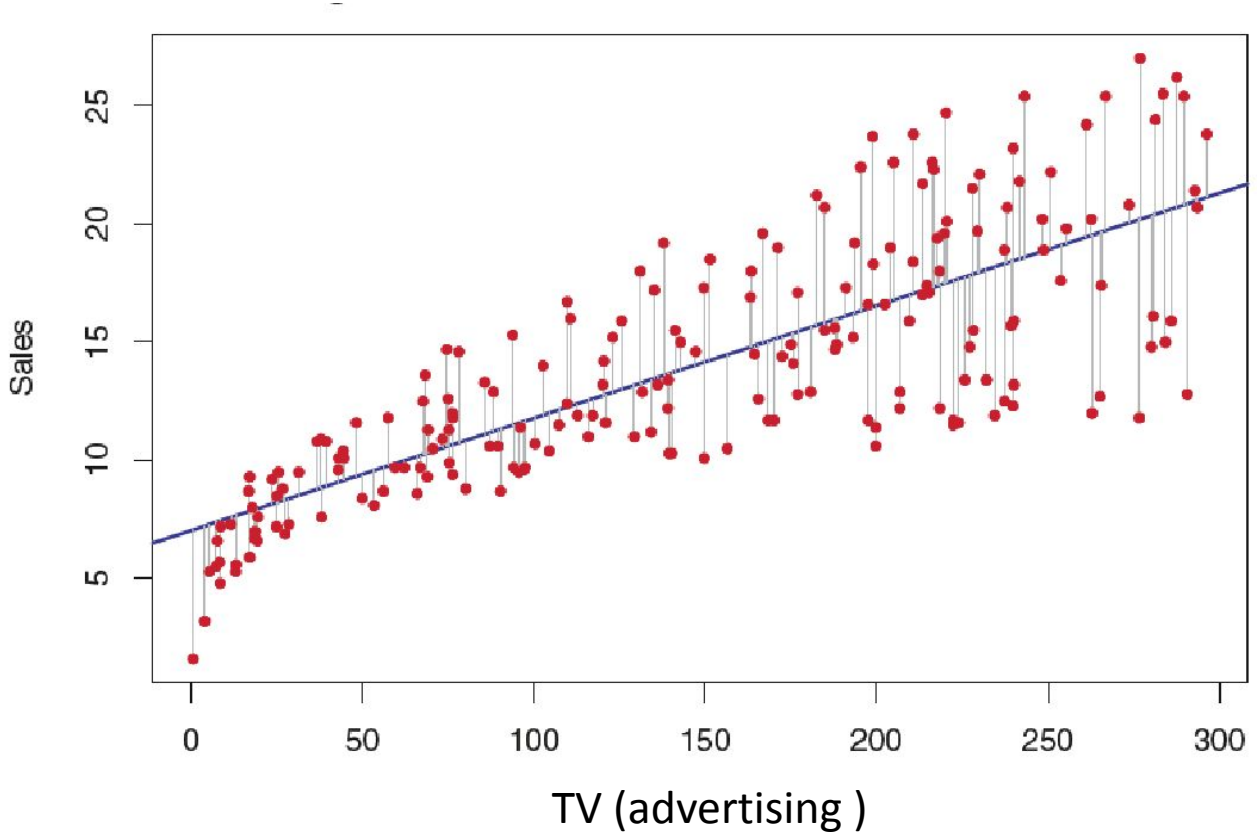


FIGURE 3.1. *For the* `Advertising` *data, the least squares fit for the regression of* `sales` *onto* `TV` *is shown. The fit is found by minimizing the sum of squared errors. Each grey line segment represents an error, and the fit makes a compromise by averaging their squares. In this case a linear fit captures the essence of the relationship, although it is somewhat deficient in the left of the plot.*

Linear Regression

Let $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$ be the prediction for Y based on the ith value of X. Then $e_i = y_i - \hat{y}_i$ represents the ith *residual*—this is the difference between the ith observed response value and the ith response value that is predicted by our linear model. We define the *residual sum of squares* (RSS) as

$$\text{RSS} = e_1^2 + e_2^2 + \cdots + e_n^2,$$

or equivalently as

$$\text{RSS} = (y_1 - \hat{\beta}_0 - \hat{\beta}_1 x_1)^2 + (y_2 - \hat{\beta}_0 - \hat{\beta}_1 x_2)^2 + \ldots + (y_n - \hat{\beta}_0 - \hat{\beta}_1 x_n)^2. \quad (3.3)$$

The least squares approach chooses $\hat{\beta}_0$ and $\hat{\beta}_1$ to minimize the RSS. Using some calculus, one can show that the minimizers are

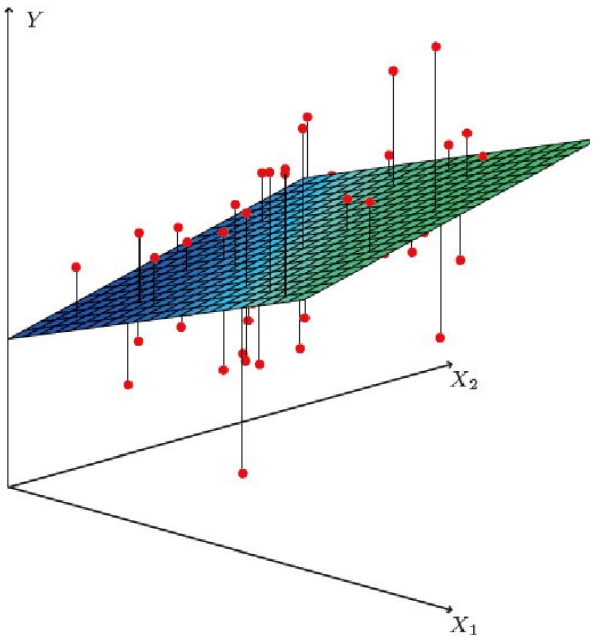
$$\begin{aligned} \hat{\beta}_1 &= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}, \\ \hat{\beta}_0 &= \bar{y} - \hat{\beta}_1 \bar{x}, \end{aligned} \quad (3.4)$$

where $\bar{y} \equiv \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\bar{x} \equiv \frac{1}{n}\sum_{i=1}^{n} x_i$ are the sample means. In other words, (3.4) defines the *least squares coefficient estimates* for simple linear regression.

Linear Regression

- Multiple linear regression considers multiple predictors:

$$Y = \beta_0 + \beta_1 \times X_1 + \beta_2 * X_2 + \varepsilon$$



An Introduction to Statistical Learning, Chapter 3

Linear Regression

- Multiple linear regression considers multiple predictors:

$$Y = \beta_0 + \beta_1 \times X_1$$

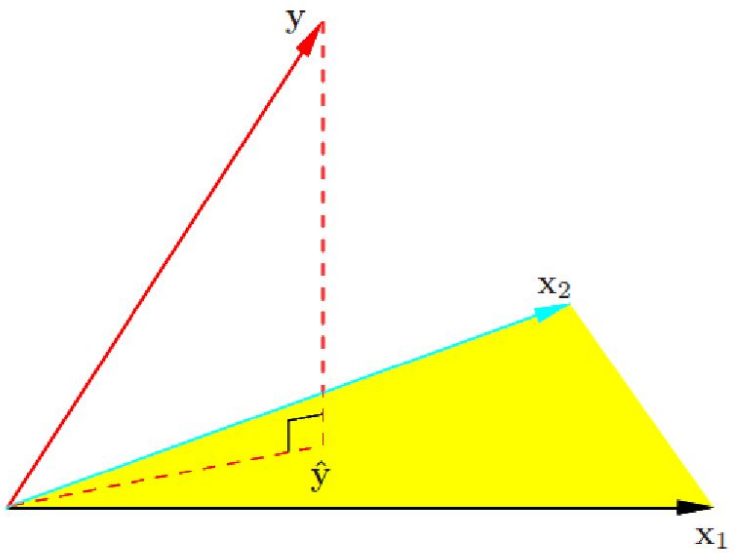


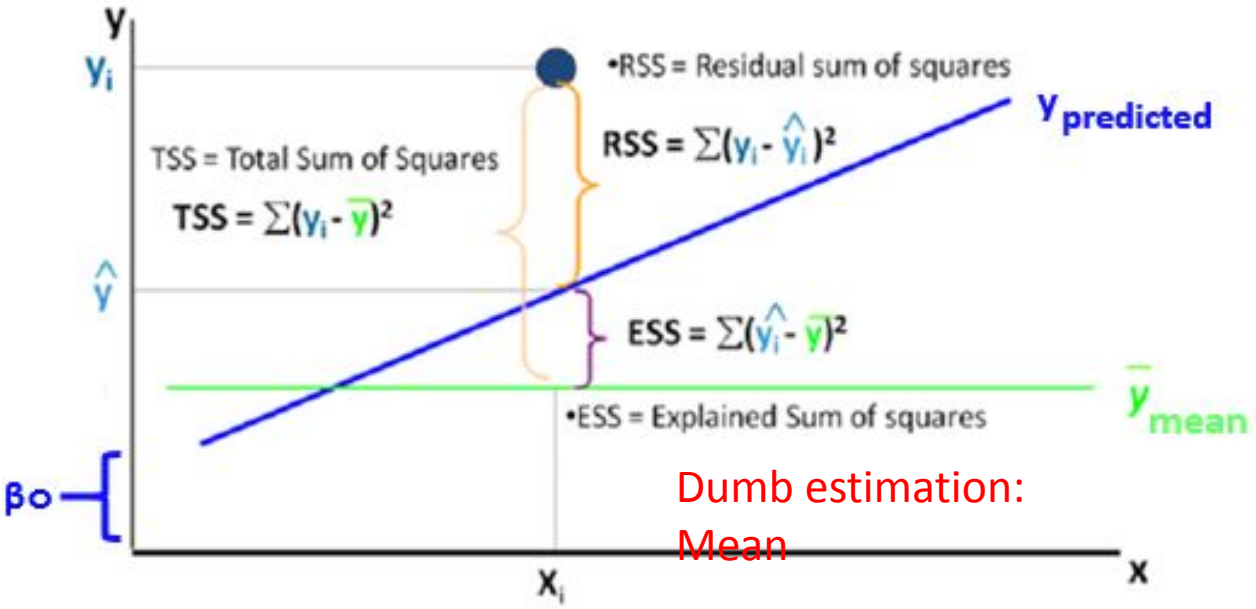
FIGURE 3.2. *The N-dimensional geometry of least squares regression with two predictors. The outcome vector* $\mathbf{y}$ *is orthogonally projected onto the hyperplane spanned by the input vectors* $\mathbf{x}_1$ *and* $\mathbf{x}_2$*. The projection* $\hat{\mathbf{y}}$ *represents the vector of the least squares predictions*

An Introduction to Statistical Learning, Chapter 3

Linear Regression

How different are our predicted values from real values?

How different is it from dumb estimation of mean (or variability of Y)?

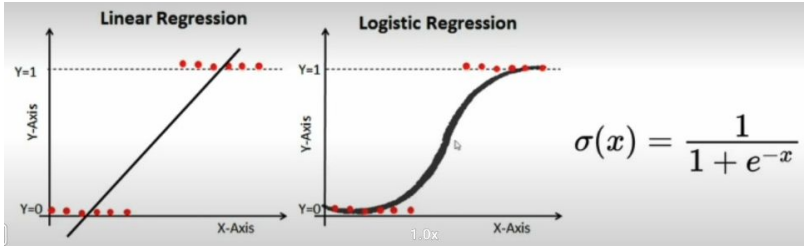


$R^2 = (TSS - RSS)/TSS = 1 - RSS/TSS$

(Proportion of variability in Y that can be explained using X)

https://www.hackerearth.com/blog/developers/beginners-guide-regression-analysis-plot-interpretations/

Logistic Regression



- Rather than modeling this response ***Y*** directly, logistic regression models the *probability* that ***Y*** belongs to a particular category
- For the Default data, logistic regression models the probability of default
- For example, the probability of default given balance can be written as **Pr(default = Yes|balance)**
- For example, one might predict **default = Yes** for any individual for whom ***p*(balance) > 0.5**

Logistic Regression

- Logistic regression w/ *logistic function (with a single predictor)*

$$p(X) = \frac{1}{1 + e^{-(\beta_0+\beta_1 X)}}$$

Logistic (or sigmoid) function: probability of Y=1 (e.g., Default = Yes) given X (e.g., Balance)

Likelihood function

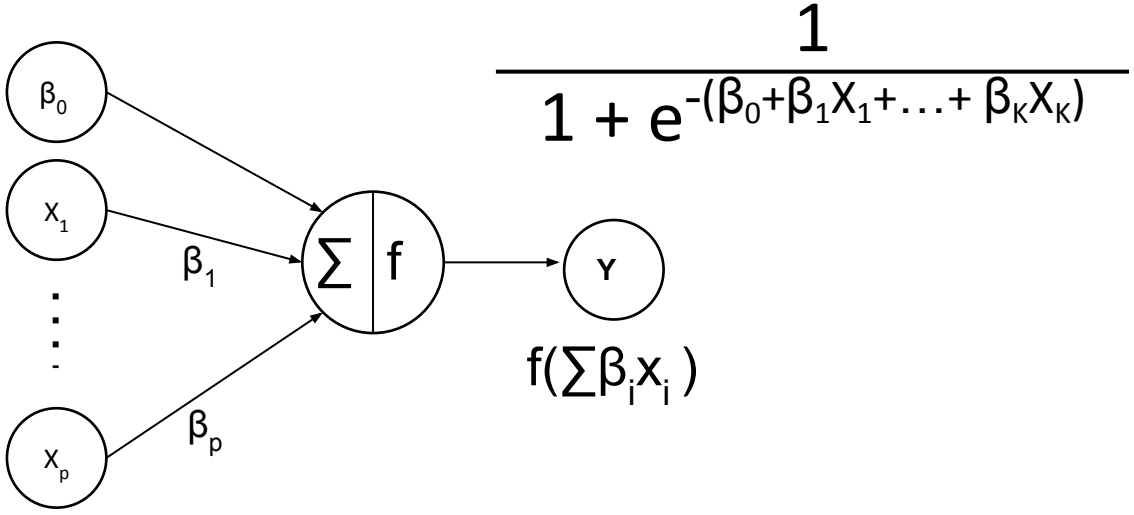
$$\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'}))$$

y_i=1 (default = Yes), $y_{i'}$=0 (default = No)

Logistic regression aims to find β_0 and β_1 via maximizing this likelihood function

Logistic Regression

- Multiple logistic regression (now w/ K predictors)
  - Very similar to a simple neural network (sigmoid - will see later)

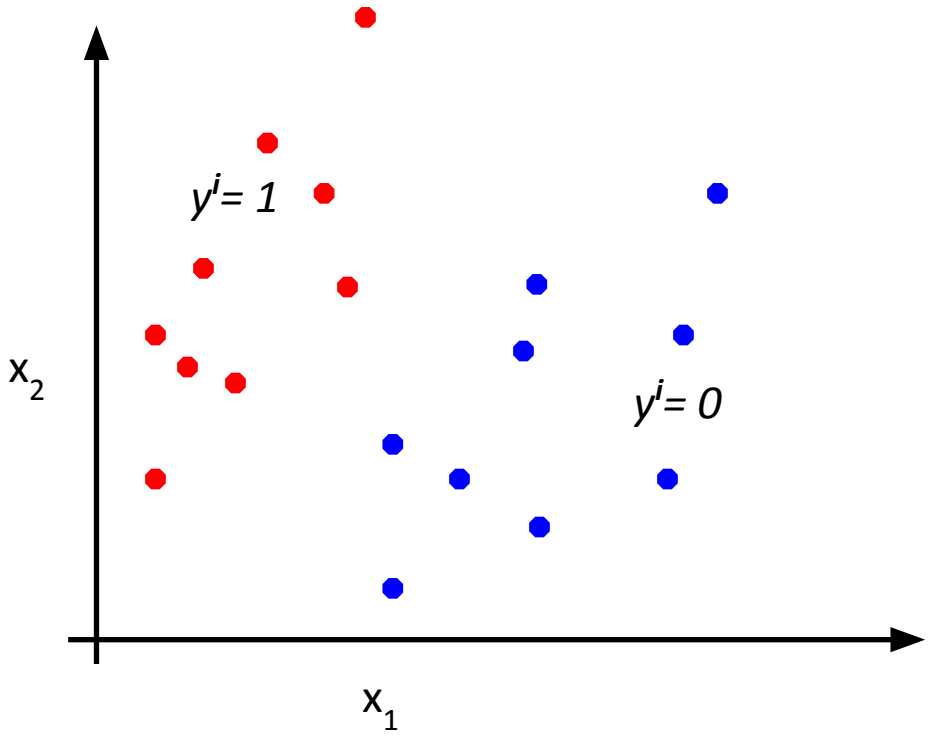


Overview

- Machine learning algorithms:
  - Decision Tree
  - kNN
  - Linear /logistic Regression
  - SVM
- Ensemble learning: bagging, boosting
- Model selection and regularization
  - Feature selection, shrinkage, dimension reduction
- Model validation: cross-validation, metrics
- Practical issues: Hyperparameter tuning, feature scaling, data augmentation, imbalanced dataset, high feature dimensions

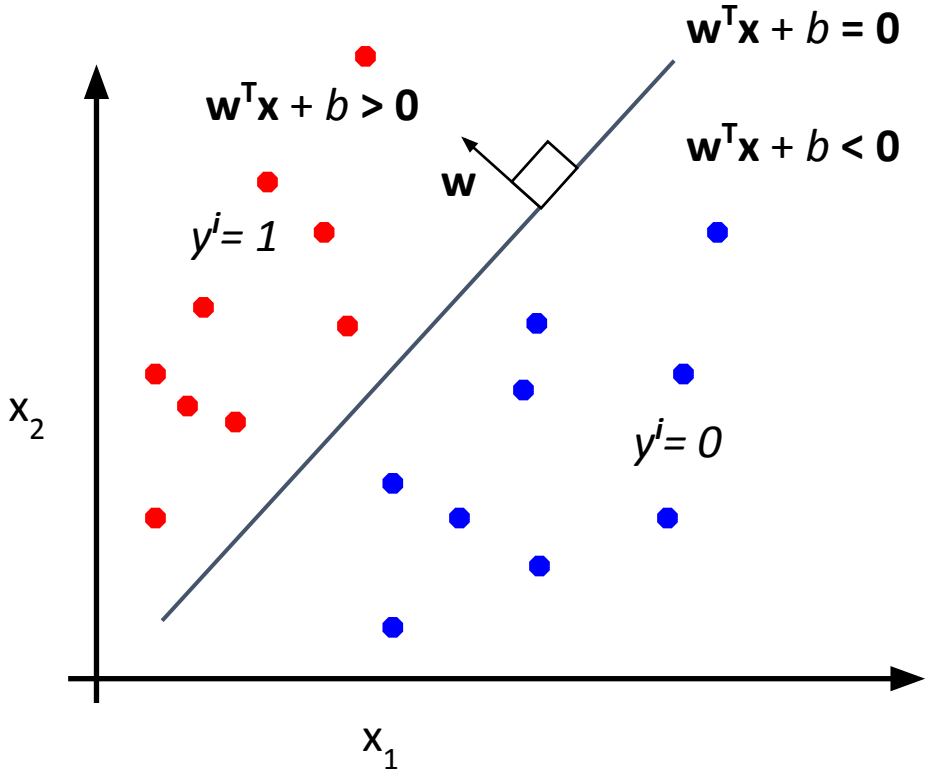
Support Vector Machine

- Binary classification can be viewed as the task of separating classes in feature space



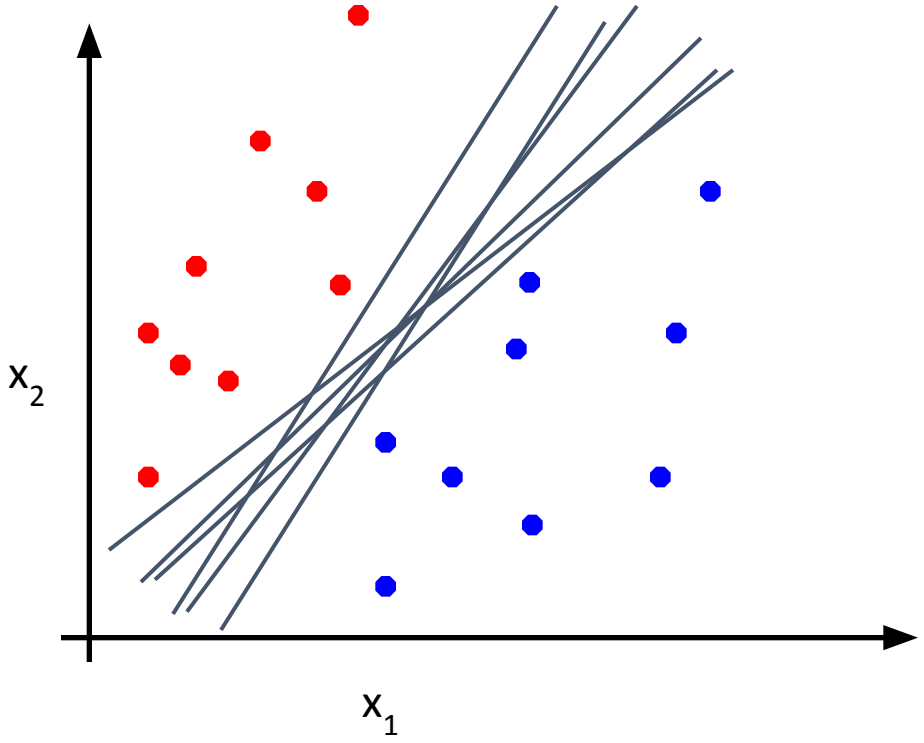
Support Vector Machine

- Binary classification can be viewed as the task of separating classes in feature space



Support Vector Machine

- Which of the linear separators is optimal?



Support Vector Machine

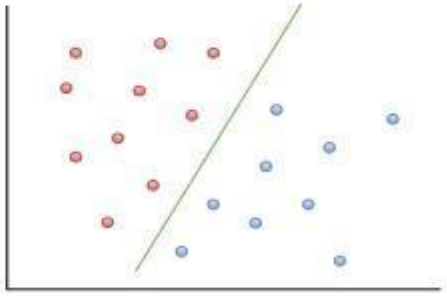
- Which of the linear separators is optimal?

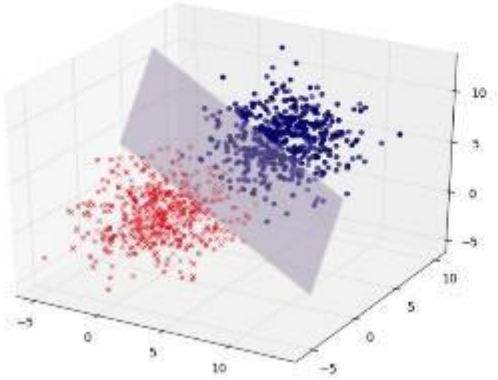
$$y = ax + b$$

Line

$$\mathbf{w}^T\mathbf{x} = 0$$

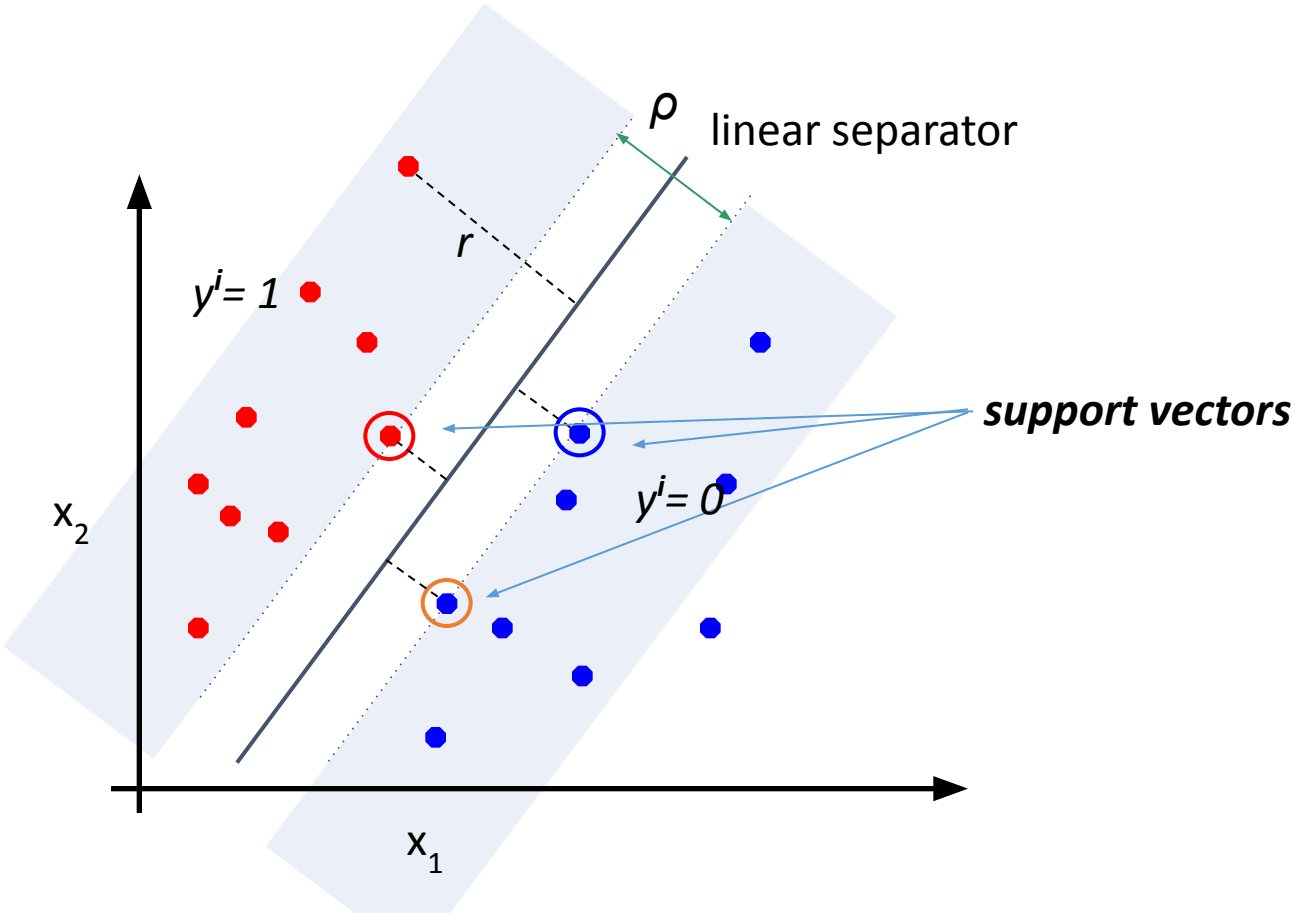
Hyperplane





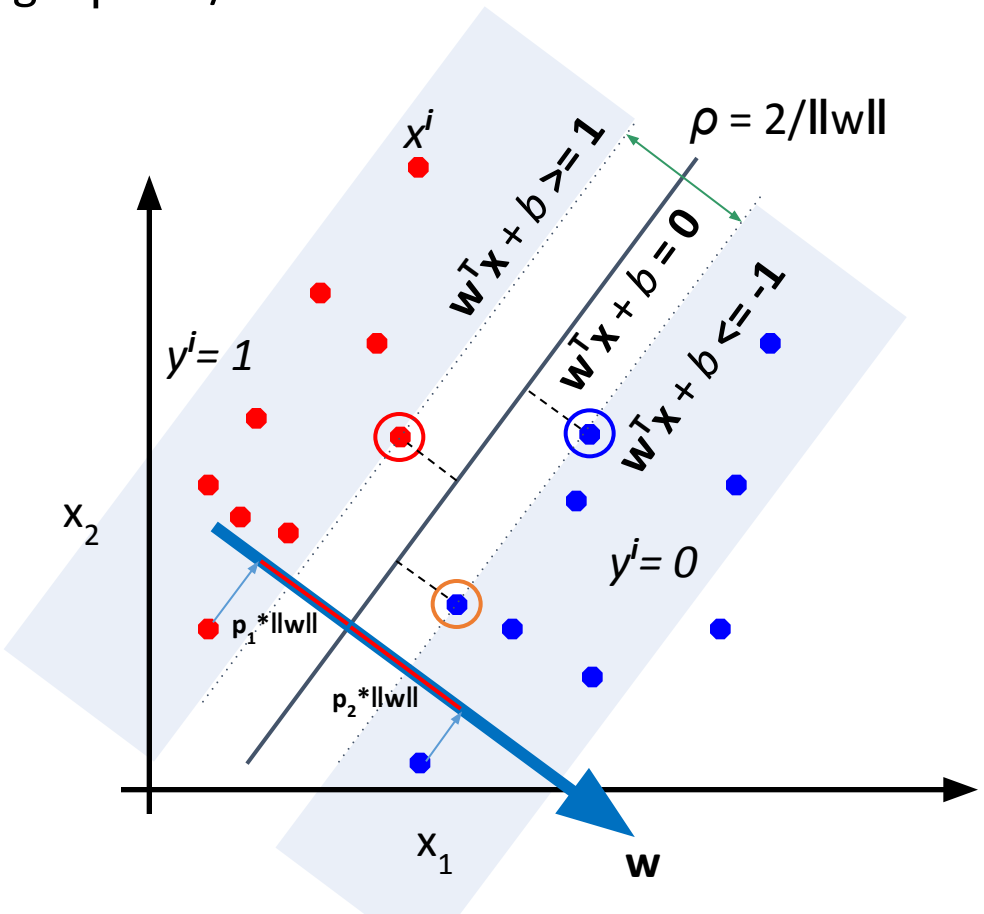
Support Vector Machine

- Examples closest to the **hyperplane** are ***support vectors***.
- ***Margin*** ρ of the separator is the distance between support vectors.



Support Vector Machine

- SVM finds the maximum-margin hyperplane
  - Find direction "w" that maximize "margin" p_i (= minimize can ||w||)
  - Max margin ρ = 2/||w||



min ½ $\|w\|^2$

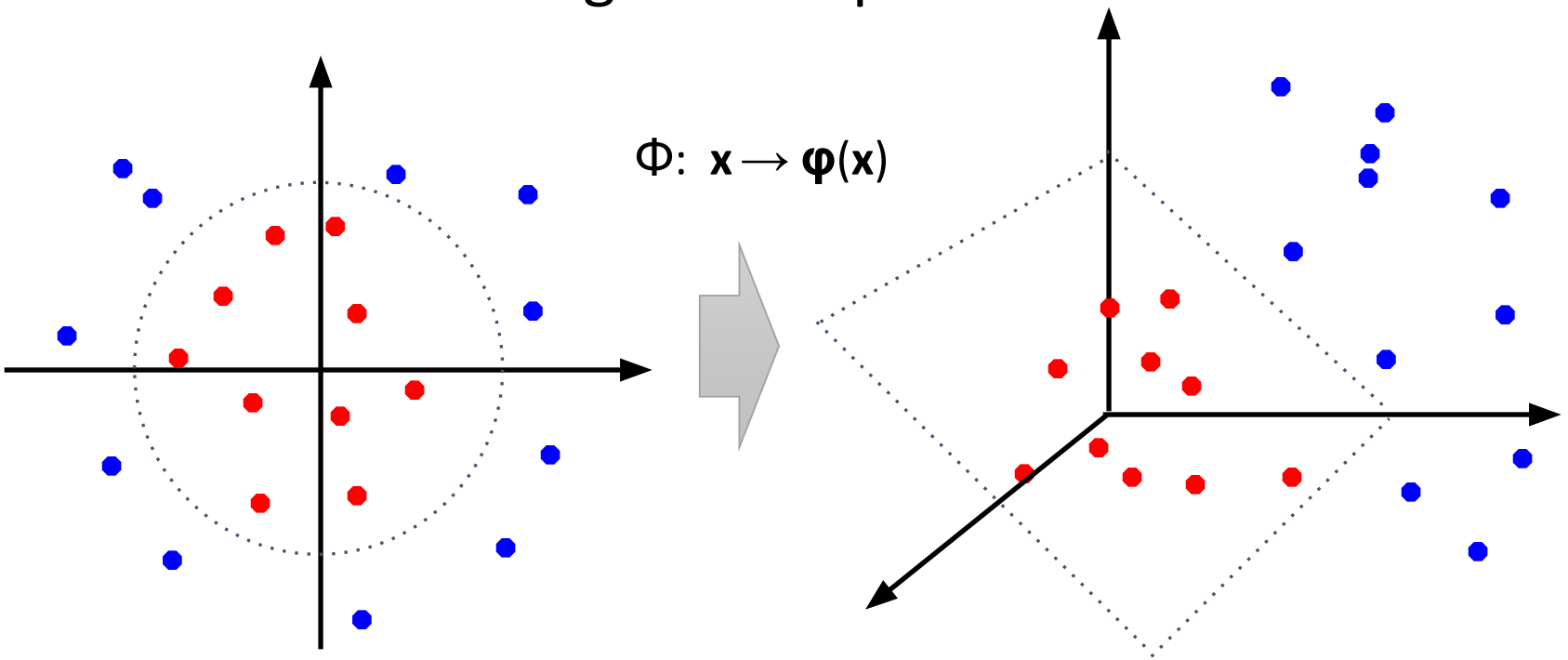
s.t.

$w^T x^i + b >= 1$ if $y^i = 1$

$w^T x^i + b <= -1$ if $y^i = 0$

Support Vector Machine

- Nonlinear SVM: original feature space can be mapped to some higher-dimensional feature space where the training set is separable



Support Vector Machine

- The linear classifier relies on inner product between vectors $K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^{\mathsf{T}}\mathbf{x}_j$
- If every data point is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \varphi(\mathbf{x})$, the inner product becomes:

$$K(\mathbf{x}_i,\mathbf{x}_j)= \boldsymbol{\varphi}(\mathbf{x}_i)^{\mathsf{T}}\boldsymbol{\varphi}(\mathbf{x}_j)$$

- Here, a kernel function $K(\mathbf{x}_i,\mathbf{x}_j)$ implicitly maps data to a high-dimensional space
- Popular nonlinear functions:
  - **Gaussian (radial-basis function)**, Polynomial

Tom Mitchell, Machine Learning McGraw Hill, 1997

SVM: Summary

- SVM is to find maximum-margin hyperplane
- Nonlinear SVM: original feature space can be mapped to some higher-dimensional feature space where the training set is separable
- SVM implementation: SMO, LibSVM, LibLinear, etc.

Overview

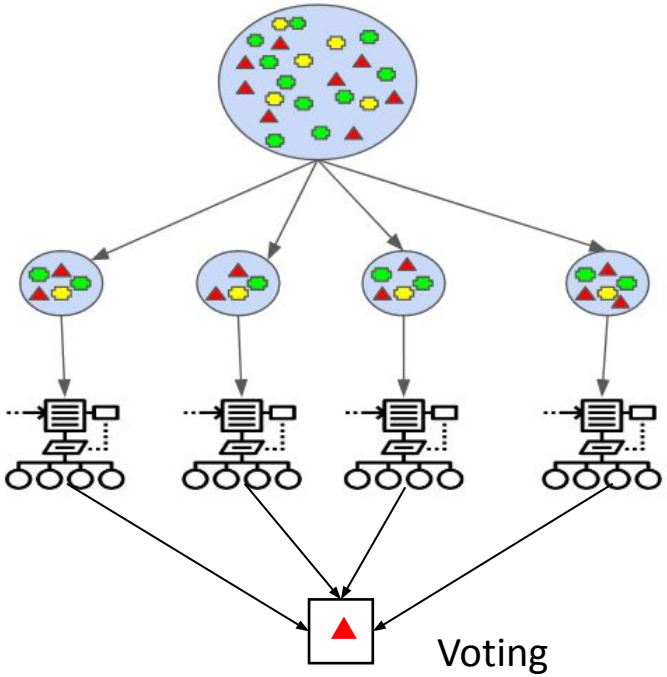
- Machine learning algorithms:
  - Decision Tree
  - kNN
  - Linear/logistic Regression
  - SVM
- **Ensemble learning: bagging, boosting**
- Model selection and regularization
  - Feature selection, shrinkage, dimension reduction
- Model validation: cross-validation, metrics
- Practical issues: Hyperparameter tuning, feature scaling, data augmentation, imbalanced dataset, high feature dimensions

Ensemble Learning

- So far, we have only discussed individual classifiers, i.e., how to build them and use them
- Can we combine multiple classifiers (of a same kind) to produce a better classifier? Yes, sometimes
  - Bagging / Randomization
  - Boosting
  - (Known as “resampling” techniques)
- When multiple classifiers of different kinds are mixed? We call it “***stacking***”

Ensemble Learning: Bagging

- **B**ootstrap **Agg**regat**ing** = Bagging



Resampling

- Given: set D containing *m* training examples
- Create a sample S[i] of D by drawing *m* examples at random with replacement from D
- S[i] of size m: expected to leave out 0.37 of examples from D

Model Training

- Create k bootstrap samples S[1], S[2], …, S[k]
- Build a distinct classifier on each S[i] to produce k classifiers, using the same learning algorithm.

Classification

- Classify each new instance by voting of the k classifiers (equal weights)

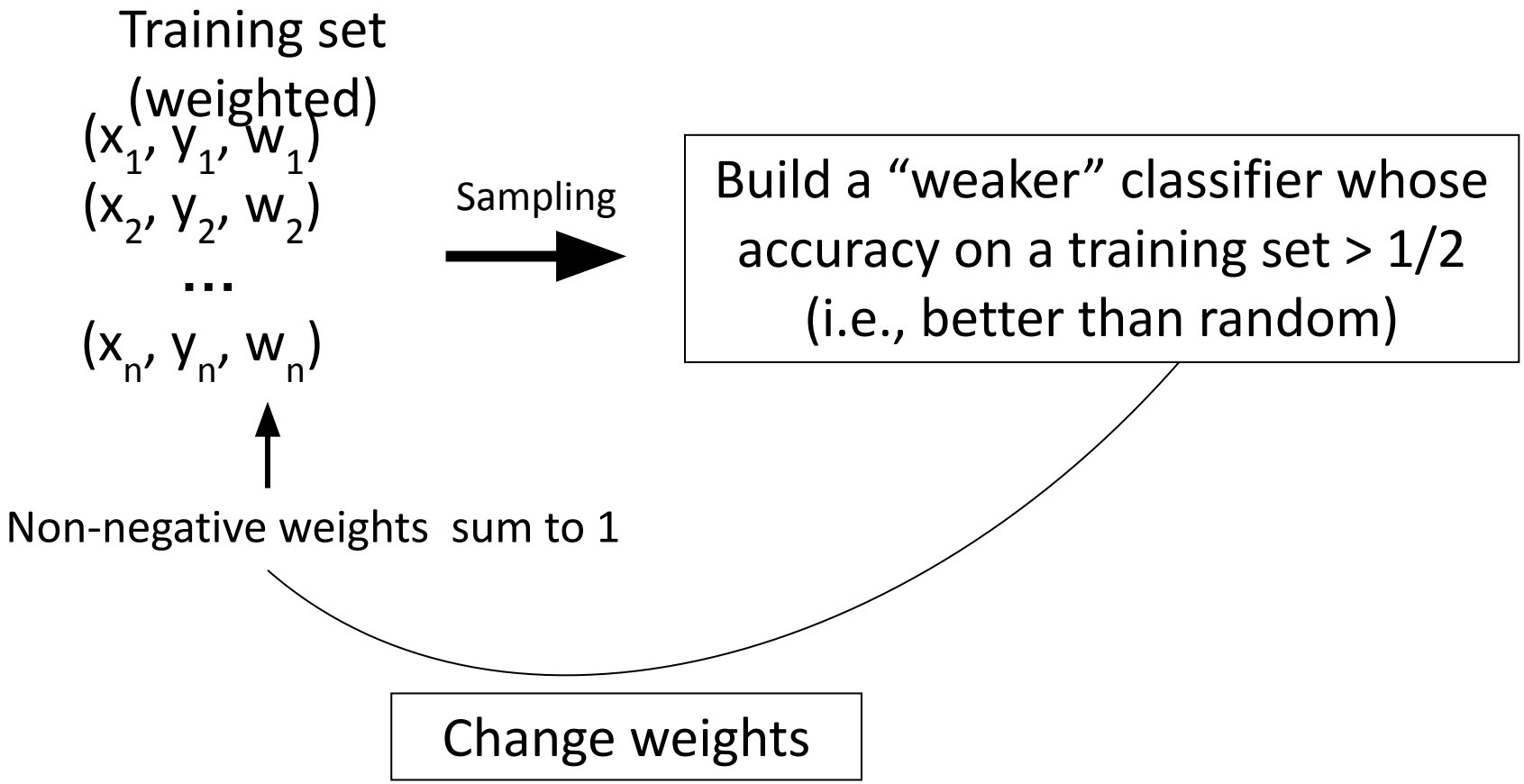
Ensemble Learning: Bagging

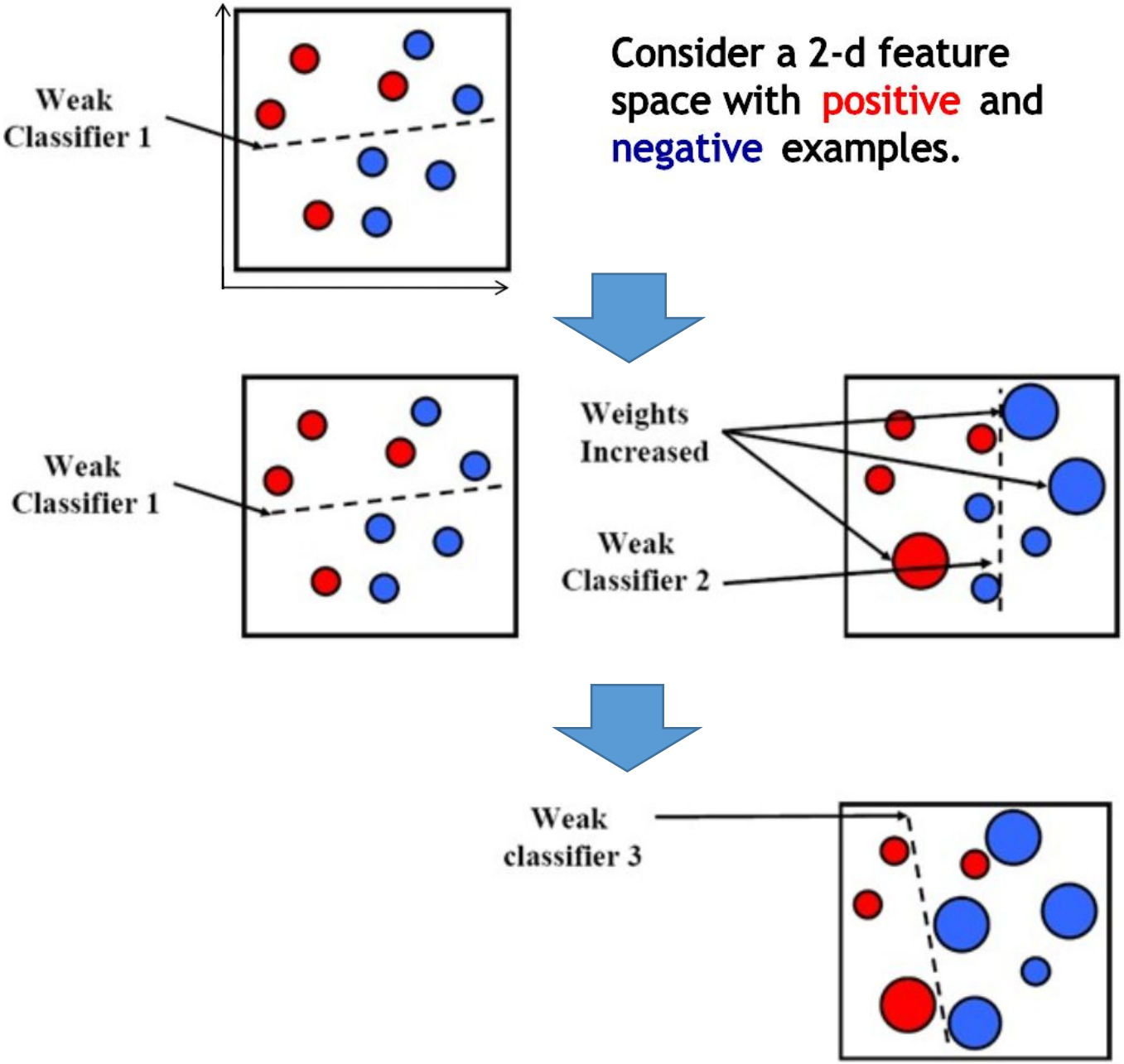
- When does it help?
- When a learner is **unstable**
  - **Small change** to training set causes **large change** in the output classifier
  - True for decision trees and neural networks; not true for k-nearest neighbor, naïve Bayesian, class association rules
- Experimentally, bagging can help substantially for unstable learners, may somewhat degrade results for stable learners

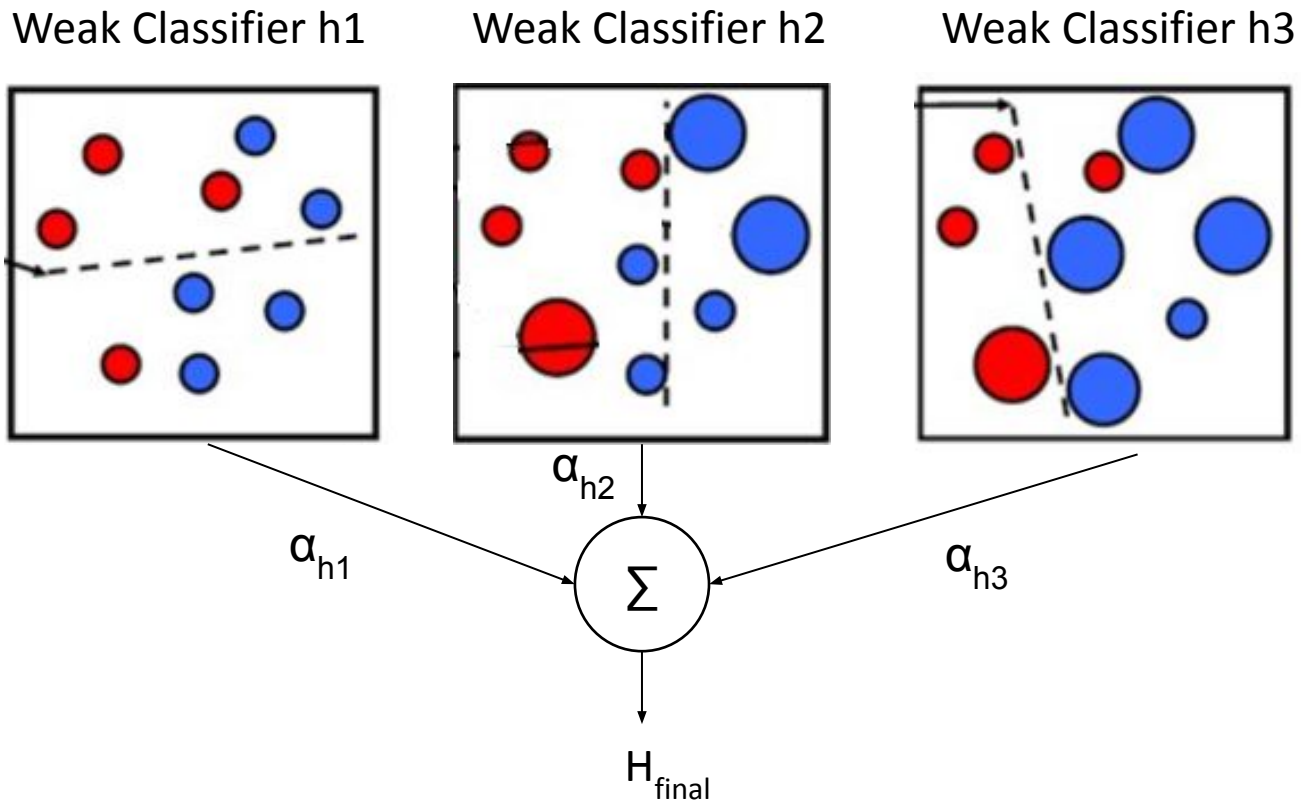
Ensemble Learning: Boosting

- A family of methods; AdaBoost is most widely used (Freund & Schapire, 1996)
- AdaBoost: Training
  - Produce a sequence of classifiers (the same base learner)
  - Each classifier is dependent on the previous one, and **focuses on the previous one's errors**
  - Examples that are incorrectly predicted in previous classifiers are given higher weights
- AdaBoost: Testing
  - For a test case, the results of the series of classifiers are combined to determine the final class of the test case (weighted average vote based on each classifier's error rate)

Ensemble Learning: Boosting




Weak
Classifier 1
Consider a 2-d feature space with positive and negative examples.
Weak
Classifier 1
Weights
Increased
Weak
Classifier 2
Weak
classifier 3


Weak Classifier h1
Weak Classifier h2
Weak Classifier h3
α_{h2}
α_{h1}
Σ
α_{h3}
H_{final}

Ensemble Learning: Boosting

- Just like AdaBoost, **Gradient Boosting** works by sequentially adding predictors to an ensemble, each one correcting its predecessor
- However, instead of tweaking the instance weights at every iteration like AdaBoost does, this method tries to fit the new predictor to the *residual errors* made by the previous predictor
- A simple regression example using Decision Trees as the base predictors; i.e., *Gradient Tree Boosting*, or *Gradient Boosted Regression Trees* (*GBRT*)
- Optimized implementation of Gradient Boosting: *XGBoost* (Extreme Gradient Boosting)

Gradient Boosted Regression Trees (*GBRT*)

Ensemble Learning: Boosting

- The actual performance of boosting depends on the data and the base learner
- Boosting seems to be susceptible to noise
  - When the number of **outliners** is very large, the emphasis placed on the hard examples (incorrectly classified examples) can hurt the performance
- Other boosting methods: Random Forests, Gradient Boosting
- More information:
  - Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions, 2010
  - https://www.slideshare.net/IntuitInc/strata-2013-how-to-create-predictive-models-in-r-using-ensembles

Overview

- Machine learning algorithms:
  - Decision Tree
  - kNN
  - Linear/logistic Regression
- Ensemble learning: bagging, boosting
- **Model selection and regularization**
  - Feature selection, shrinkage, dimension reduction
- Model validation: cross-validation, metrics
- Practical issues: Hyperparameter tuning, Feature scaling, data augmentation, imbalanced dataset, high feature dimensions

Model selection & Regularization

- How can we better fit the model that can yield better prediction accuracy and model interpretability?
  - Prediction accuracy:
    - If we have too many features (as opposed to the number of samples), variance is high (suffering from **overfitting**)
    - May want to constrain or shrink estimated coefficients to reduce the variance
  - Model interpretability:
    - Including "irrelevant variables" may result in **unnecessary complexity**
    - Easy to interpret the model by removing irrelevant variables (e.g., by making coefficients zero)

An Introduction to Statistical Learning, Chapter 6

Model selection & Regularization

- Feature/Subset Selection
  - Identifying a subset of the *p* predictors that we believe to be related to the response.
  - A model is trained based on the reduced set of variables
- Regularization/Shrinkage
  - Fitting a model involving all *p* predictors, but the estimated coefficients are shrunken towards zero
  - This shrinkage (also known as regularization) has the effect of reducing variance
  - Depending on what type of shrinkage is performed, some of the coefficients may be estimated to be exactly zero. Hence, shrinkage methods can also perform variable selection
- Dimension Reduction
  - This approach involves projecting the *p* predictors into a M-dimensional subspace, where M < p
  - This is achieved by computing M different linear combinations, or projections, of the variables, which are used for model training

An Introduction to Statistical Learning, Chapter 6

Model selection & Regularization

- Feature selection
  - Feature selection is also called “attribute selection”
  - Belongs to “supervised” methods (it’s supervised as “class” info is used for selection)
- Dimension reduction (or data projection)
  - Principle component analysis (PCA) / Auto-encoders
  - Belongs to “unsupervised” methods
- Regularization (shrinkage)
  - Adding regularization terms in the objective functions: e.g., $\lambda\sum|\beta_i|$ (L1 norm), $\lambda\sum\beta_i^2$ (L2 norm)
  - Dropouts & early stopping in neural networks
  - Belongs to “supervised” methods

Regularization

- L1/L2 regularization
- Linear regression examples:
  - Ridge regression (L2 norm)

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

  - Lasso regression (L1 norm)

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

An Introduction to Statistical Learning, Chapter 6

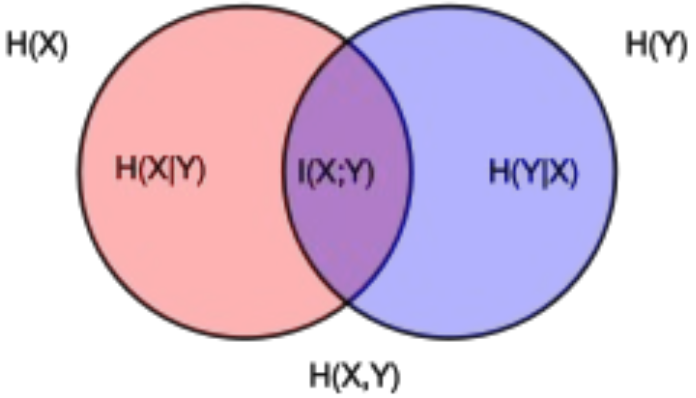
Feature Selection

Filter methods	Wrapper methods
Generic set of methods which do not incorporate a **specific machine learning algorithm.**	Evaluates on a **specific machine learning algorithm** to find optimal features.
Much **faster** compared to Wrapper methods in terms of time complexity	**High computation time** for a dataset with many features
Less prone to **over-fitting**	High chances of **over-fitting** because it involves training of machine learning models with different combination of features
Examples – **Correlation, Chi-Square test, ANOVA, Information gain** etc.	Examples - **Forward Selection, Backward elimination, Stepwise selection** etc.

- Filter vs. wrapper
  - **Filter**: select features "only" based on general characteristics of the train data
  - **Wrapper**: use a target learning algorithm to estimate the worth of feature subsets
- Rank vs. subset selection
  - Rank: evaluate (and hence rank) individual features (e.g., Information Gain, Relief)
  - Subset selection: evaluate (and hence rank) subsets of features (e.g., **correlation**-based feature selection: CFS, **consistency**-based subset selection: CSS)
    - Various "subset search methods" can be employed: e.g., forward, backward, best first, greedy stepwise, genetic search, rank-search, etc.

Benchmarking Attribute Selection Techniques for Discrete Class Data Mining, Mark A. Hall, Georey Holmes, IEEE TKDE 2003

Feature Selection



- Information gain
  - Each feature A_i is assigned a score based on information gain between itself and the class: $H(C) - H(C|A_i)$ (very similar to that of decision tree)
    - H(C) – entropy of the class (overall data set)
    - $H(C|A_i)$ – entropy of the class given A_i
    - This metric is also known as "mutual information"
  - "Removing features with low variance" can be considered as removing features with "low information gain"
- CFS (Correlation-based Feature Selection)
  - Take into account the usefulness of individual features for predicting the class along with the level of inter-correlation among them
  - Give high scores to subsets containing features that are highly correlated with the class and have low inter-correlation with each other

From Entropy to Information Gain

Entropy $H(X)$ of a random variable X

$$H(X) = -\sum_{i=1}^{n} P(X=i)\log_2 P(X=i)$$

Specific conditional entropy $H(X|Y=v)$ of X given $Y=v$:

$$H(X|Y=v) = -\sum_{i=1}^{n} P(X=i|Y=v)\log_2 P(X=i|Y=v)$$

Conditional entropy $H(X|Y)$ of X given Y :

$$H(X|Y) = \sum_{v \in values(Y)} P(Y=v)H(X|Y=v)$$

Mututal information (aka Information Gain) of X and Y :

$$I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

http://www.cs.cmu.edu/~tom/10601_fall2012/slides/DTreesAndOverfitting-9-4-2012.pdf

Dimension Reduction

- Principal Component Analysis (PCA)
- t-SNE (t-Distributed Stochastic Neighbor Embedding)
- Autoencoder

Overview

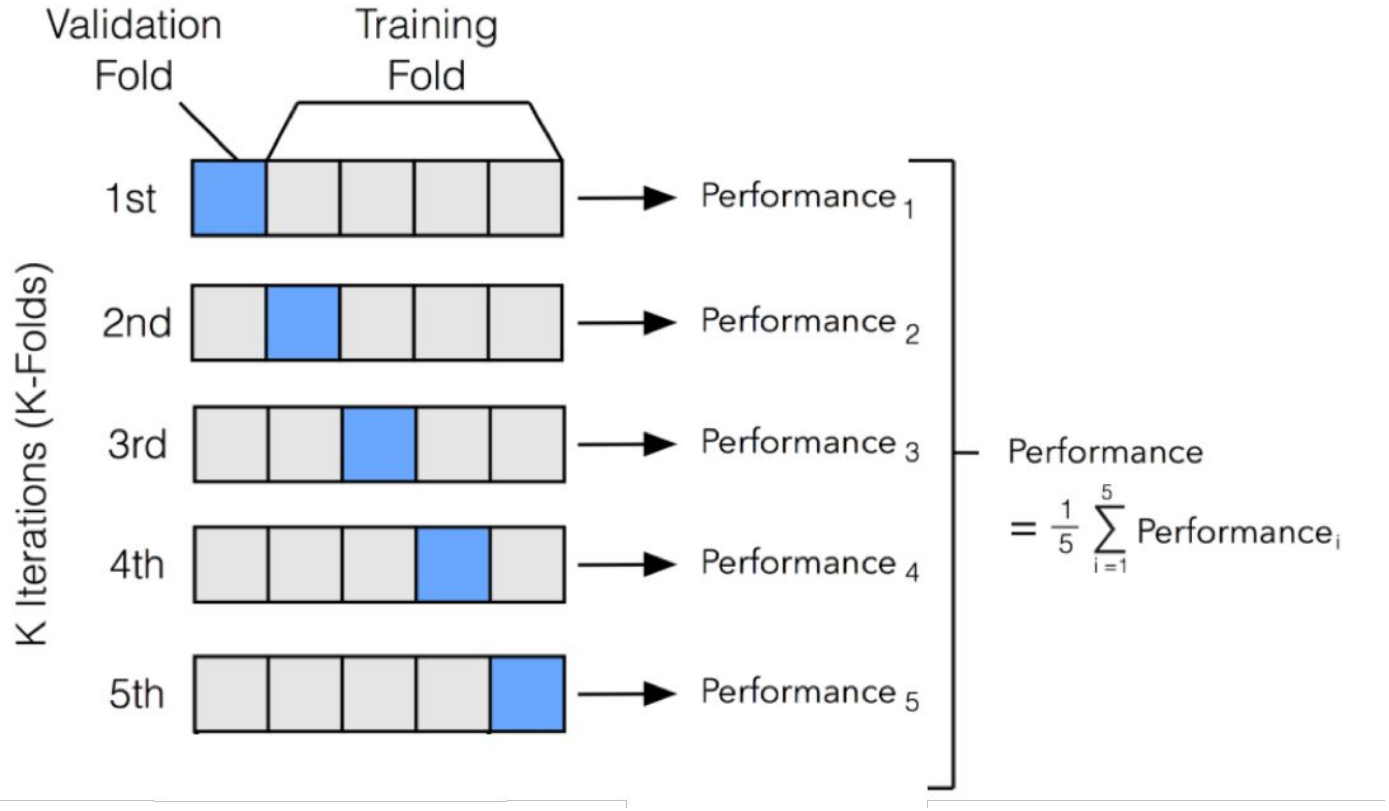
- Machine learning algorithms:
  - Decision Tree
  - kNN
  - Linear/logistic Regression
- Ensemble learning: bagging, boosting
- Model selection and regularization
  - Feature selection, shrinkage, dimension reduction
- **Model validation: evaluation, metrics**
- Practical issues: Hyperparameter tuning, Feature scaling, data augmentation, imbalanced dataset, high feature dimensions

Model Evaluation

- **K-fold cross-validation**
  - The data is divided randomly into k parts in which the class is represented in approximately the same proportions as in the full dataset (random sampling assumption)
  - Each part is held out in turn and the learning scheme trained on the remaining k-1 parts; then its error rate is calculated on the holdout set
  - Thus learning procedure is executed a total of k times on different training sets.
  - Finally, the k error estimates are averaged to yield an overall error estimate

Model Evaluation

- 5-fold cross-validation



http://ethen8181.github.io/machine-learning/model_selection/model_selection.html

Model Evaluation

- **Hold-out** validation
  - Simple split of a dataset: e.g., 80% for training, 20% for validation
- **K-fold** cross-validation
  - The data is divided **randomly** into k parts, and each part is held out in turn and the learning scheme trained on the remaining k-1 parts
  - Extreme case: **Leave-one-out cross-validation (aka** *n*-fold cross-validation) - *n* is # instances in the dataset
  - Assumption: individual samples are independent of one another (meaning that re-ordering shouldn't influence performance)
- **LOSO**: Leave-one-subject-out cross-validation
  - If there are ***n*** participants, a model is trained with n-1 users, and then is tested with 1 left out participant (known as "subject or user independent modeling")
  - But if a data set is limited (i.e., only a few users), applying LOSO would be difficult mainly due to large sample variability (i.e., sample size is simply too small). In this case, researchers usually use "**user-dependent models"** with hold-out validation (Hammerla, Plotz, 2015)

Let's (not) stick together: pairwise similarity biases cross-validation in activity recognition, Nils Y. Hammerla, Thomas Plötz, ACM Ubicomp 2015 https://dl.acm.org/citation.cfm?id=2807551

Model Evaluation

- klearn.model_selection: Model Selection

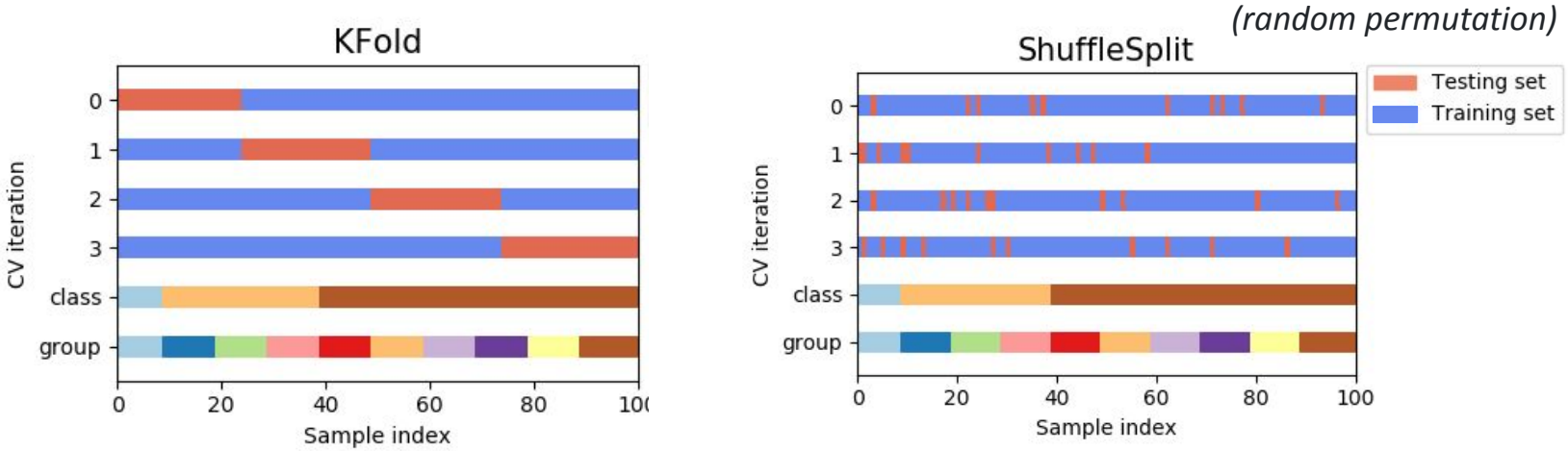
Splitter Classes	
model_selection.GroupKFold([n_splits])	K-fold iterator variant with non-overlapping groups.
model_selection.GroupShuffleSplit([...])	Shuffle-Group(s)-Out cross-validation iterator
model_selection.KFold([n_splits, shuffle, ...])	K-Folds cross-validator
model_selection.LeaveOneGroupOut	Leave One Group Out cross-validator
model_selection.LeavePGroupsOut(n_groups)	Leave P Group(s) Out cross-validator
model_selection.LeaveOneOut	Leave-One-Out cross-validator
model_selection.LeavePOut(p)	Leave-P-Out cross-validator
model_selection.PredefinedSplit(test_fold)	Predefined split cross-validator
model_selection.RepeatedKFold([n_splits, ...])	Repeated K-Fold cross validator.
model_selection.RepeatedStratifiedKFold([...])	Repeated Stratified K-Fold cross validator.
model_selection.ShuffleSplit([n_splits, ...])	Random permutation cross-validator
model_selection.StratifiedKFold([n_splits, ...])	Stratified K-Folds cross-validator
model_selection.StratifiedShuffleSplit([...])	Stratified ShuffleSplit cross-validator
model_selection.TimeSeriesSplit([n_splits, ...])	Time Series cross-validator

Splitter Functions	
model_selection.check_cv([cv, y, classifier])	Input checker utility for building a cross-validator
model_selection.train_test_split(*arrays, ...)	Split arrays or matrices into random train and test subsets

Model Evaluation
sklearn.model_selection

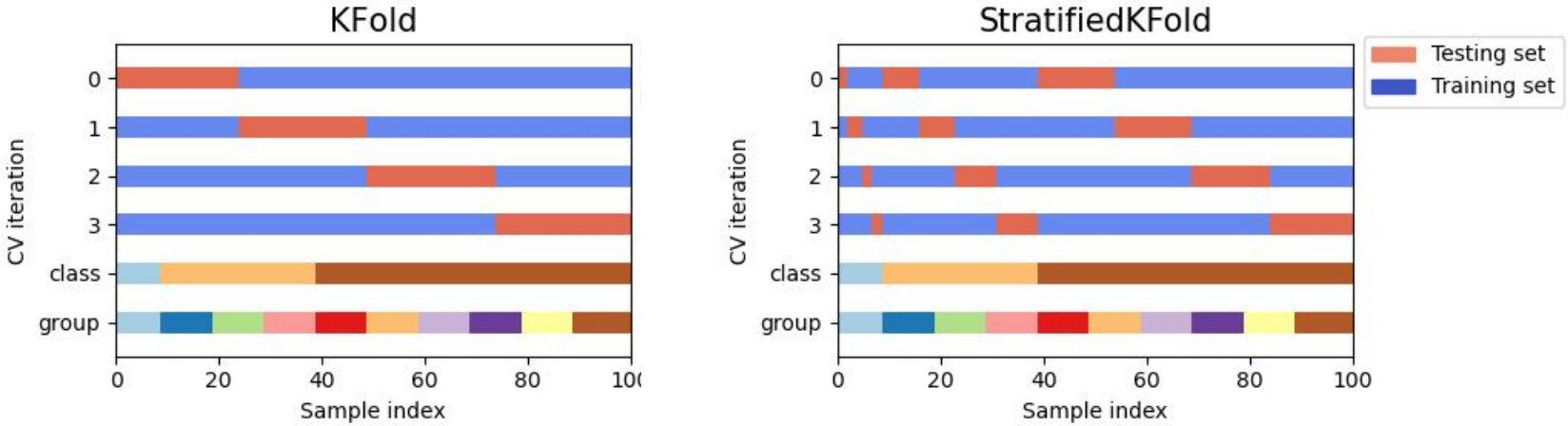
- **Kfold** (n_splits=4, random_state=12, **shuffle=True**)
  - n_splits - Number of folds (default = 5)
  - shuffle - before splitting it into batches (but order within a split is preserved)
  - **random_state** - Set an int for reproducible output across multiple function calls or None (= default) for randomized results
- **ShuffleSplit** (n_splits=4, random_state=12, **test_size=0.25, train_size=None**)
  - n_splits = Number of re-shuffling & splitting iterations
  - test_size = proportion of the dataset to include in the test split

(random permutation)



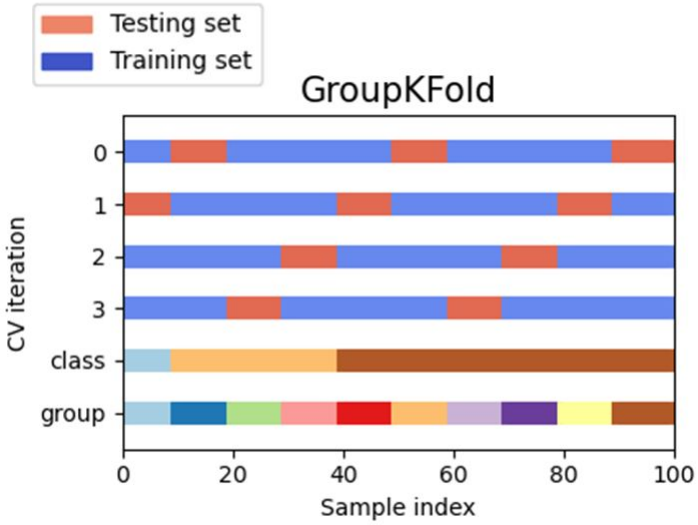
Model Evaluation

- **StratifiedKFold** (n_splits=2, random_state=12, shuffle=False)
  - Some classification problems can exhibit a large imbalance in the distribution of the target classes: for instance there could be several times more negative samples than positive samples
  - StratifiedKFold - Relative class frequencies are approximately preserved in each train and validation fold
  - Must specify “shuffle = True” to shuffle each class’s samples before splitting into batches



Model Evaluation

- GroupKFold(n_splits=n)
  - Person as a group; each group has multiple samples
  - Same person(i.e., group) is never in both test and training



Imagine you have three subjects, each with an associated number from 1 to 3:

```
>>> from sklearn.model_selection import GroupKFold

>>> X = [0.1, 0.2, 2.2, 2.4, 2.3, 4.55, 5.8, 8.8, 9, 10]
>>> y = ["a", "b", "b", "b", "c", "c", "c", "d", "d", "d"]
>>> groups = [1, 1, 1, 2, 2, 2, 3, 3, 3, 3]

>>> gkf = GroupKFold(n_splits=3)
>>> for train, test in gkf.split(X, y, groups=groups):
...     print("%s %s" % (train, test))
[0 1 2 3 4 5] [6 7 8 9]
[0 1 2 6 7 8 9] [3 4 5]
[3 4 5 6 7 8 9] [0 1 2]
```

Model Evaluation

- **Wrong way to do cross-validation**
  1. Screen the predictors: find a subset of "good" predictors that show fairly strong (univariate) correlation with the class labels
  2. Using just this subset of predictors, build a multivariate classifier
  3. Use cross-validation to estimate the unknown tuning parameters and to estimate the prediction error of the final model

The Elements of Statistical Learning, Chapter 7.10.2

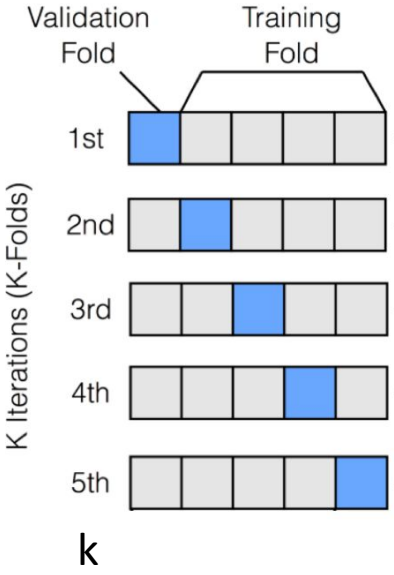
Model Evaluation

- Wrong way to do cross-validation: “contrived scenario”
  - N = 50 samples in two equal-sized classes
  - p = 5000 quantitative predictors (standard Gaussian) that are independent of the class labels
  - The true (test) error rate of any classifier is 50%
  - Applied previous strategies
    - Choosing in step (1) the 100 predictors having highest correlation with the class labels, and then using a 1-nearest neighbor classifier, based on just these 100 predictors, in step (2).
    - Over 50 simulations from this setting, the average CV error rate was 3%. This is far lower than the true error rate of 50%
- Why?
  - The problem is that the predictors have an **unfair advantage**, as they were chosen in step (1) on the basis of all of the samples
  - Leaving samples out after the variables have been selected does not correctly mimic the application of the classifier to a completely independent test set, since these predictors “**have already seen**” the left out samples

The Elements of Statistical Learning, Chapter 7.10.2

Model Evaluation

- Correct way to do cross-validation
  1. Divide the samples into K cross-validation folds (groups) at random
  2. For each fold k = 1, 2, . . . , K
     1. Find a subset of "good" predictors that show fairly strong (univariate) correlation with the class labels, using all of the samples **except those in fold k**
     2. Using just this subset of predictors, build a multivariate classifier, using all of the samples **except those in fold k**
     3. Use the classifier to predict the class labels for the samples in fold k

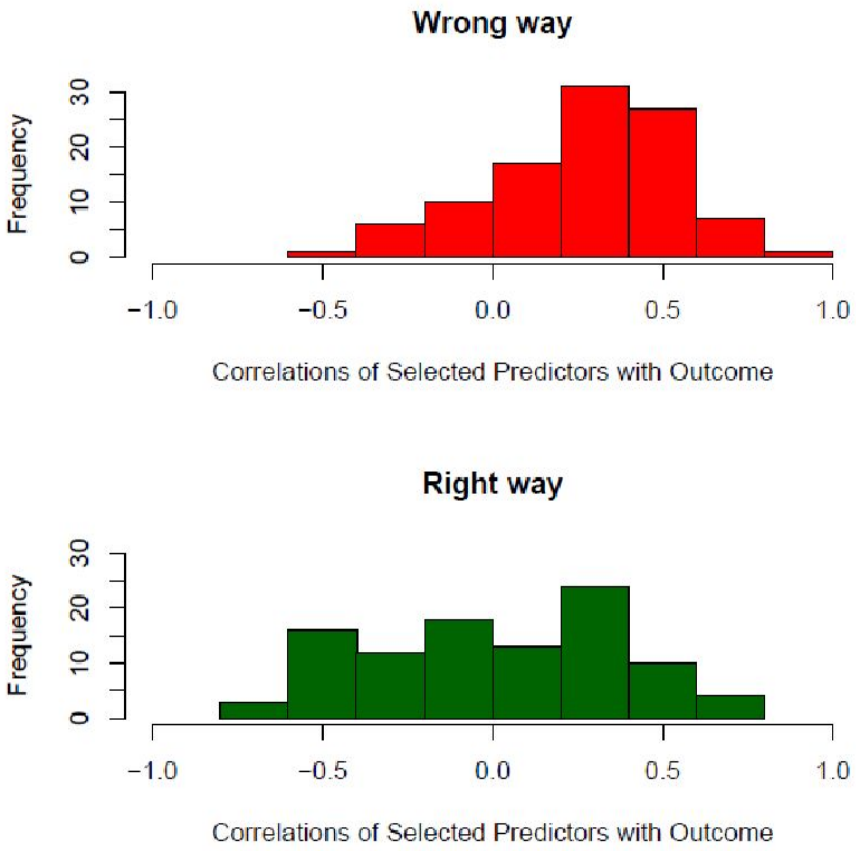


The Elements of Statistical Learning, Chapter 7.10.2

Model Evaluation

- Wrong vs. Correct way to do cross-validation

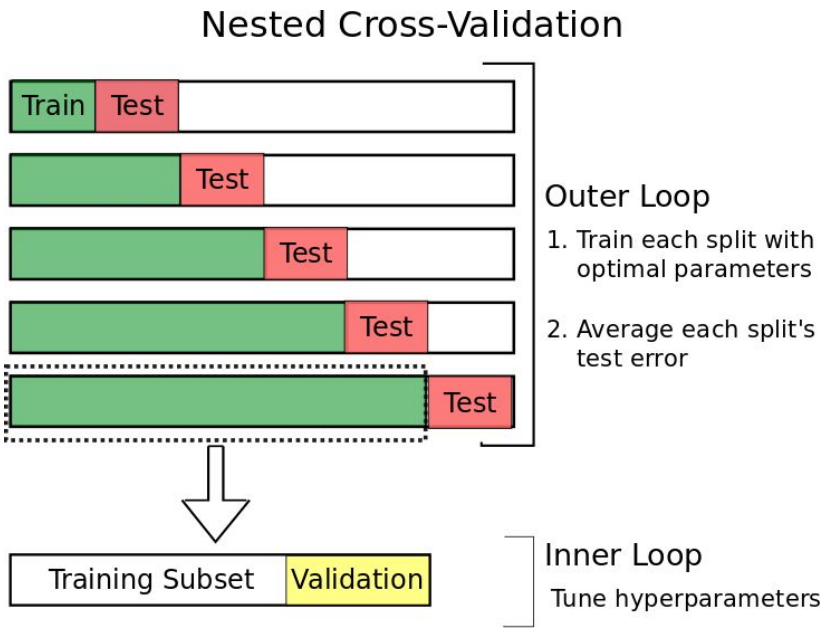
Histograms shows the correlation of class labels, in 10 randomly chosen samples, with the 100 predictors chosen using the incorrect (upper red) and correct (lower green) versions of cross-validation.



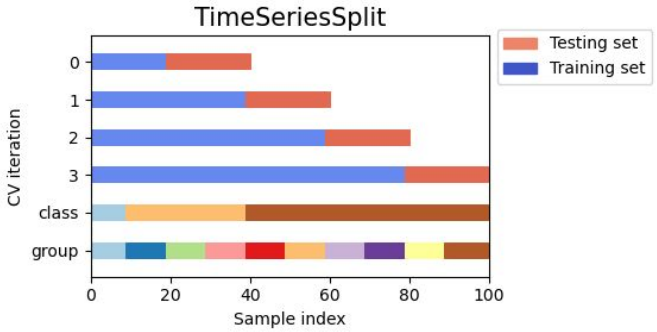
The Elements of Statistical Learning, Chapter 7.10.2

Model Evaluation

- Model evaluation for time-series data
  - Due to temporal dependency, we should avoid using k-fold, but it's recommended to use hold-out cross-validation



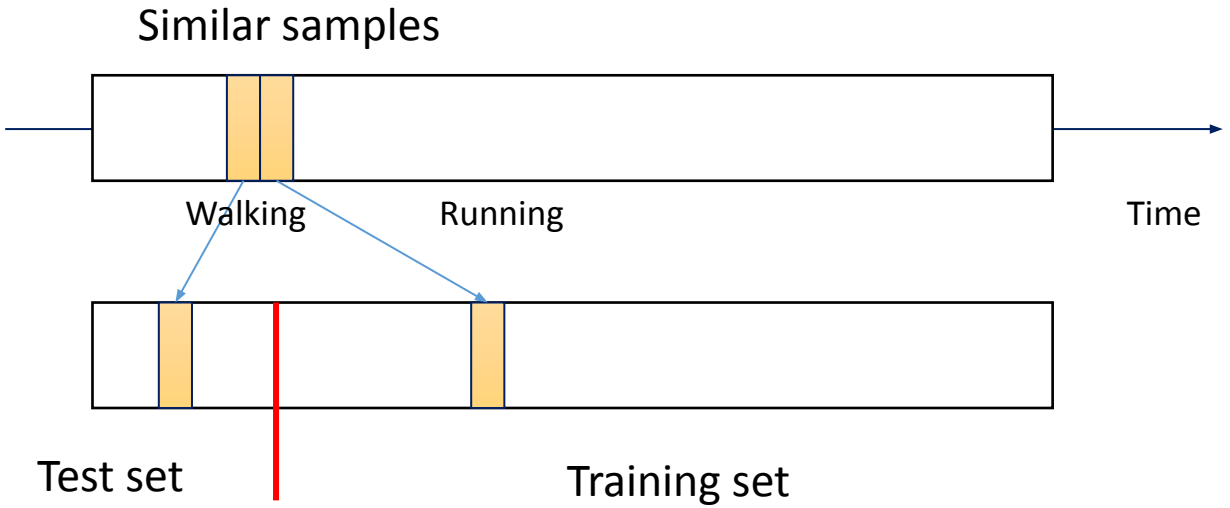
TimeSeriesSplit in sklearn



S. Varma and R. Simon. Bias in error estimation when using cross-validation for model selection. BMC Bioinformatics, 7(1):91, Feb 2006. ISSN 1471–2105. doi: 10.1186/1471–2105- 7–91 https://towardsdatascience.com/time-series-nested-cross-validation-76adba623eb9

Model Evaluation

- Dealing w/ temporal correlation
  - **Neighborhood-bias** (nearby samples likely have the same labels, i.e., they are not independent) must be considered in cross-validation (i.e., avoid placing adjacent frames in different folds)"



Let's (not) stick together: pairwise similarity biases cross-validation in activity recognition, Nils Y. Hammerla, Thomas Plötz, ACM Ubicomp 2015 https://dl.acm.org/citation.cfm?id=2807551

Model Evaluation

- Comparing performance of algorithms
  - Measures: precision, recall, F-score

(confusion matrix)

		True Class A	*True Class* Not A
Predicted Class	*A*	**True Positive (tp)**	**False Positive (fp)**
	Not A	**False Negative (fn)**	**True Negative (tn)**

N1	**N2**
N3	**N4**

- **Precision** = tp / (tp + fp)
- **Recall** = tp / (tp + fn) (aka, sensitivity)
- **Accuracy** = (tp + tn) / (tp+tn+fp+fn)
  = correctly predicted / all cases (N)
- **F1-score**: harmonic mean of precision & recall

$$F_1 = \left(\frac{\text{recall}^{-1} + \text{precision}^{-1}}{2} \right)^{-1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Model Evaluation

- **Precision** = tp / (tp + fp)
- **Recall** = tp / (tp + fn) (aka, sensitivity)
- **Accuracy** = (tp + tn) / (tp+tn+fp+fn)
  = correctly predicted / all cases (N)
- **F1-score**: harmonic mean of precision & recall

$$F_1 = \left(\frac{\text{recall}^{-1} + \text{precision}^{-1}}{2} \right)^{-1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

- What happens if we flip positive & negative?

Predicted

	A	*Not A*
A	True Positive (tp)	False Positive (fp)
Not A	False Negative (fn)	True Negative (tn)

True Class

	A	*Not A*
A	True Negative (tn)	False Negative (fn)
Not A	False Positive (fp)	True Positive (tp)

True Class

N1	N2
N3	N4

Precision = N1 / (N1 + N2)
Recall = N1 / (N1 + N3)

Precision = N4 / (N3 + N4)
Recall = N4 / (N2 + N4)

Depending on we consider which label is a "positive" label, precision/recall/f-measure values are very different => As one solution, we can report the average values of these two cases (aka. "macro-average")

- What happens if we have multiple classes?
  - One vs. all others for evaluation

		True/Actual		
		Cat ()	Fish ()	Hen ()
Predicted	Cat ()	4	6	3
	Fish ()	1	2	0
	Hen ()	1	2	6

https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1

		True/Actual		
		Cat (🐱)	Fish (🐟)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐟)	1	2	0
	Hen (🐔)	1	2	6

Precision = 4 / (4 + 6+ 3)

Recall = 4 / (4 + 1 + 1)

		True/Actual		
		Cat (🐱)	Fish (🐟)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐟)	1	2	0
	Hen (🐔)	1	2	6

Precision = 2 / (1 + 2+ 0)

Recall = 2 / (6 + 2 + 2)

		True/Actual		
		Cat (🐱)	Fish (🐟)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐟)	1	2	0
	Hen (🐔)	1	2	6

Precision = 6 / (1 + 2+ 6)

Recall = 6 / (3 + 0 + 6)

	precision	recall	f1-score	support
Cat	0.308	0.667	0.421	6
Fish	0.667	0.200	0.308	10
Hen	0.667	0.667	0.667	9

https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1

Model Evaluation: Multi-class

- How to aggregate multiple evaluation results?
  - **Macro-averaging** gives equal weight to each class (label)
  - **Weighted-averaging** based on the number of true samples from each class (=support)
  - **Micro-averaging** gives equal weight to each classification decision (correctly predicted or not, regardless of class labels)

		True/Actual		
		Cat (🐱)	Fish (🐟)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐟)	1	2	0
	Hen (🐔)	1	2	6

Precision = 4 / (4 + 6+ 3)

Recall = 4 / (4 + 1 + 1)

		True/Actual		
		Cat (🐱)	Fish (🐟)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐟)	1	2	0
	Hen (🐔)	1	2	6

Precision = 2 / (1 + 2+ 0)

Recall = 2 / (6 + 2 + 2)

		True/Actual		
		Cat (🐱)	Fish (🐟)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐟)	1	2	0
	Hen (🐔)	1	2	6

Precision = 6 / (1 + 2+ 6)

Recall = 6 / (3 + 0 + 6)

	precision	recall	f1-score	support
Cat	0.308	0.667	0.421	6
Fish	0.667	0.200	0.308	10
Hen	0.667	0.667	0.667	9
accuracy			0.480	25
macro avg	0.547	0.511	0.465	25
weighted avg	0.581	0.480	0.464	25

- **Macro-averaging** gives equal weight to each class (label)
  - Macro-avg of precision = (0.308+0.668+0.667)/3 = 0.547
  - Macro-avg of f1-score = (0.421+0.308+0.667)/3 = 0.465

$$f_1 = \frac{2}{c} \sum_{i=1}^{c} \frac{\text{precision}_i \times \text{recall}_i}{\text{precision}_i + \text{recall}_i}$$

c = # classes

Let's (not) stick together: pairwise similarity biases cross-validation in activity recognition, Nils Y. Hammerla, Thomas Plötz, ACM Ubicomp 2015 https://dl.acm.org/citation.cfm?id=2807551

		True/Actual		
		Cat (🐱)	Fish (🐟)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐟)	1	2	0
	Hen (🐔)	1	2	6

Precision = 4 / (4 + 6+ 3)

Recall = 4 / (4 + 1 + 1)

		True/Actual		
		Cat (🐱)	Fish (🐟)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐟)	1	2	0
	Hen (🐔)	1	2	6

Precision = 2 / (1 + 2+ 0)

Recall = 2 / (6 + 2 + 2)

		True/Actual		
		Cat (🐱)	Fish (🐟)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐟)	1	2	0
	Hen (🐔)	1	2	6

Precision = 6 / (1 + 2+ 6)

Recall = 6 / (3 + 0 + 6)

	precision	recall	f1-score	support
Cat	0.308	0.667	0.421	6
Fish	0.667	0.200	0.308	10
Hen	0.667	0.667	0.667	9
accuracy			0.480	25
macro avg	0.547	0.511	0.465	25
weighted avg	0.581	0.480	0.464	25

- **Weighted-averaging** based on the number of true samples from each class
  - Weighted-avg of precision = (0.308*6+0.668*10+0.667*9)/25 = 0.581
  - Weighted-avg of f1-score = (0.421*6+0.308*10+0.667*9)/25 = 0.464

	precision	recall	f1-score	support
Cat	0.308	0.667	0.421	6
Fish	0.667	0.200	0.308	10
Hen	0.667	0.667	0.667	9
accuracy			0.480	25
macro avg	0.547	0.511	0.465	25
weighted avg	0.581	0.480	0.464	25

		True/Actual		
		Cat (🐱)	Fish (🐟)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐟)	1	2	0
	Hen (🐔)	1	2	6

- **Micro-averaging** gives equal weight to each classification decision
  - Precision = tp / (tp+fp)
  - Recall = tp / (tp+fn)
  - Accuracy = tp / N
  - **Precision = recall = f1-score = accuracy**

tp = 4 + 2 + 6 = 12

fp = fn = tn = all the other cells (6 +3 +1 +0 + 1 + 2 = 13)

Macro- and micro-averaged evaluation measures, Asch 2013

https://pdfs.semanticscholar.org/1d10/6a2730801b6210a67f7622e4d192bb309303.pdf

Model Evaluation: Baseline

- Learned model should be better than dumb models
  - Majority voting: blindly vote for the major class (known as Zero Rule, or ZeroR)
    - Use **DummyClassifier(strategy**="most_frequent") in Scikit-learn
  - Random voting: randomly vote for any class

https://stackoverflow.com/questions/29441943/what-is-the-theorical-foundation-for-scikit-learn-dummy-classifier

Data imbalance & evaluation metrics

		True class: p	True class: n
Hypothesis output	Y	TP (True Positives)	FP (False Positives)
	N	FN (False Negatives)	TN (True Negatives)
Column counts:		P_C	N_C

$$Accuracy = \frac{TP + TN}{P_C + N_C}; \quad ErrorRate = 1 - accuracy$$

Major 95%, minor 5% => majority classification : accuracy 95%

Key Issue: any metric that uses values from both columns will be inherently sensitive to dataset imbalances

$$Precision = \frac{TP}{TP + FP},$$

$$Recall = \frac{TP}{TP + FN},$$

$$F\text{-}Measure = \frac{(1+\beta)^2 \cdot Recall \cdot Precision}{\beta^2 \cdot Recall + Precision},$$

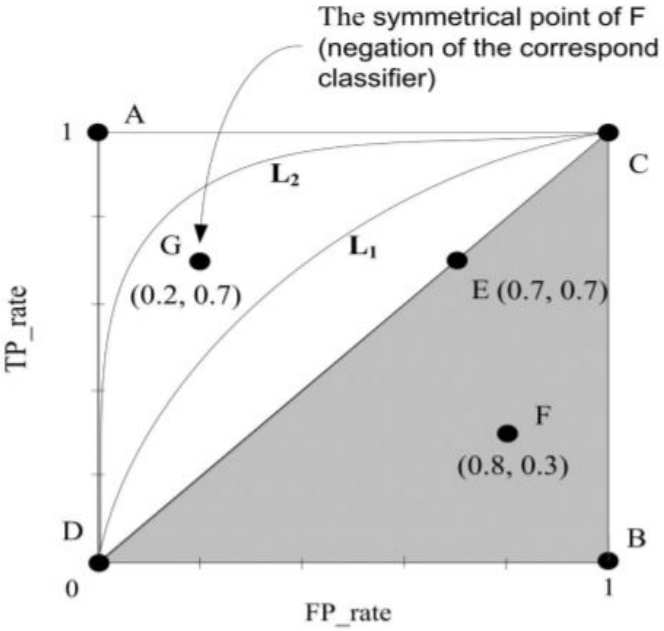
- Precision is sensitive to data distributions, while recall is not.
- But recall provides no insight to how many examples are incorrectly labeled as positive; likewise, precision cannot assert how many positive examples are labeled incorrectly.
- F-measure still suffers from limitations of precision and recall.

He and Garcia, Learning from Imbalanced Data

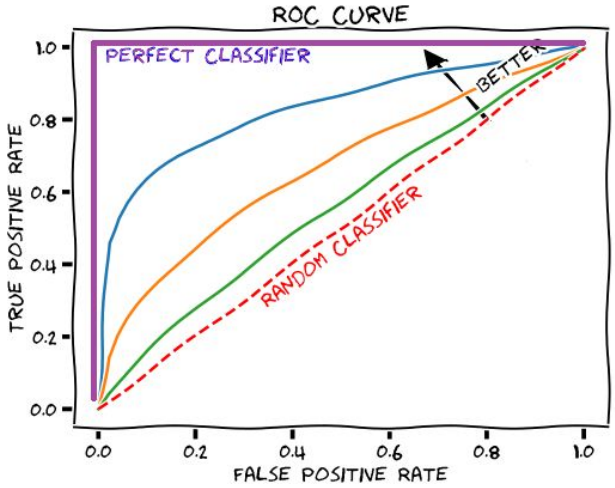
Data imbalance & evaluation metrics

Hypothesis output	True class: p	True class: n
Y	TP (True Positives)	FP (False Positives)
N	FN (False Negatives)	TN (True Negatives)
Column counts:	P_C	N_C

$$TP\_rate = \frac{TP}{P_C}; \quad FP\_rate = \frac{FP}{N_C}.$$



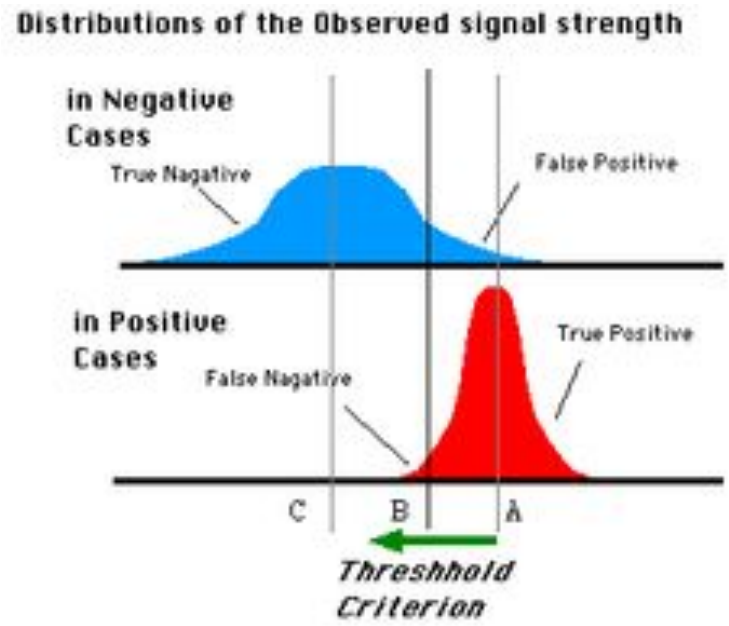
- The ROC graph is formed by plotting TP rate over FP rate, and any point in ROC space corresponds to the performance of a single classifier on a given distribution
- The ROC curve is useful because it provides a visual representation of the **relative trade-offs** between the benefits (reflected by true positives) and costs (reflected by false positives) of classification in regards to data distributions

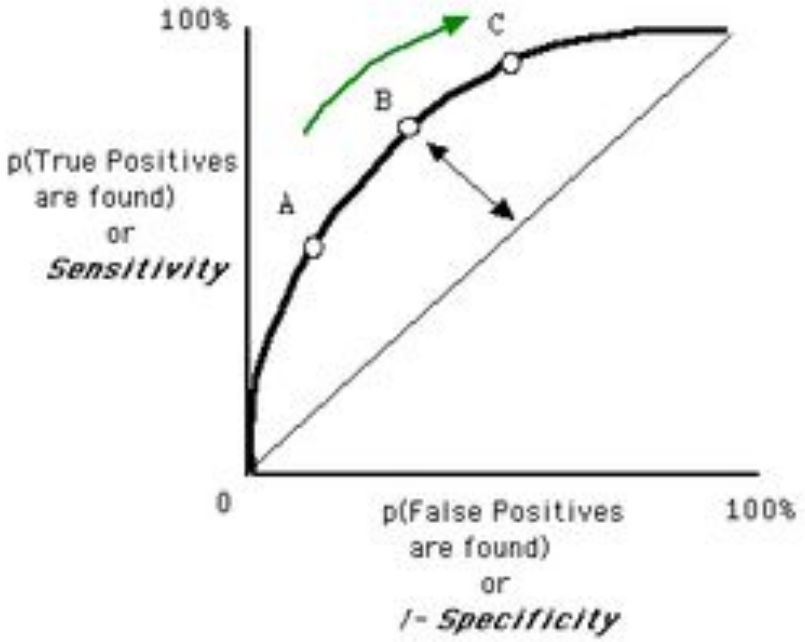


https://glassboxmedicine.com/2019/02/23/measuring-performance-auc-auroc/

He and Garcia, Learning from Imbalanced Data

Model Evaluation: ROC curves

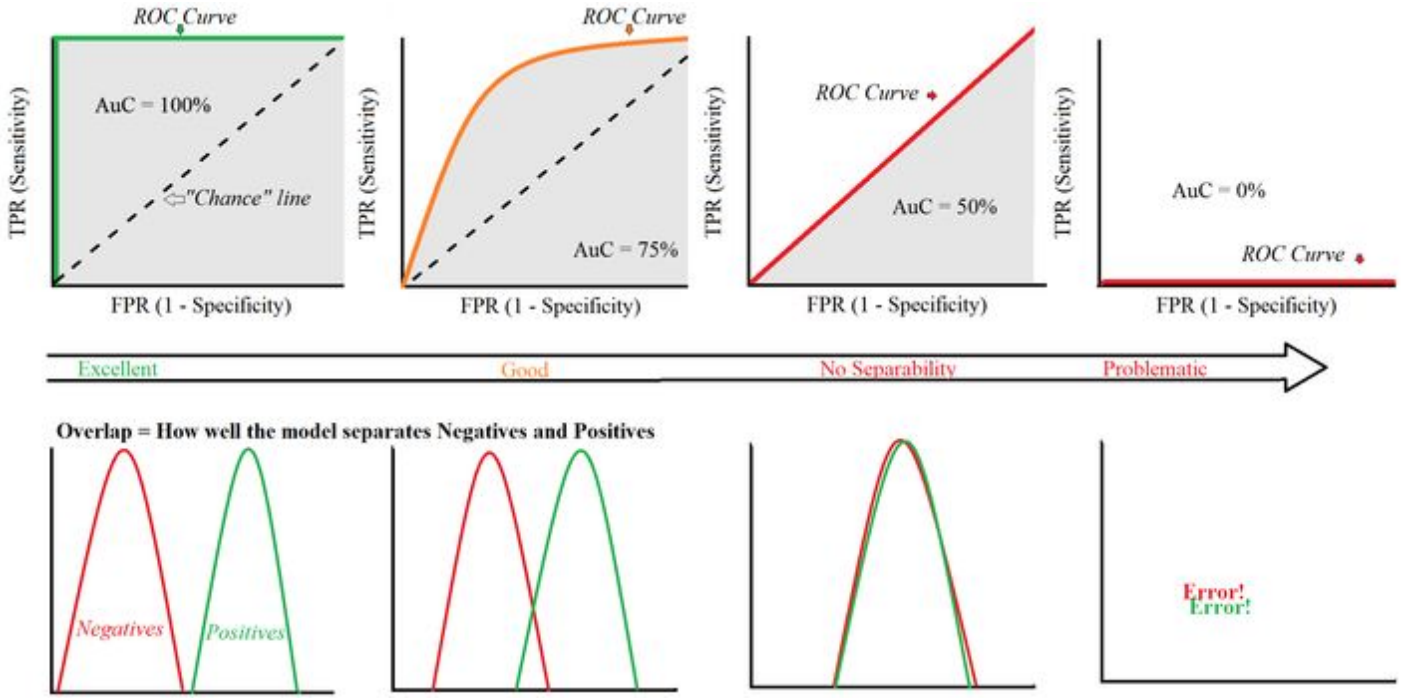




ROC is useful when dealing with a soft-type classifier (or **a ranking or scoring classifier)** (e.g., Naïve Bayes, Neural Network) where **a threshold is used to produce a discrete (binary) classifier**: if the classifier output is above the threshold, the classifier produces a Y, else a N

An introduction to ROC analysis, Tom Fawcett, 2006

Model Evaluation: AUC-ROC



- Comparing multiple classifiers using the area under the curve (AUC)
- Larger AUC means better average performance
- Yet it is still possible that a high AUC classifier could perform worse in a specific region in ROC space than a low AUC classifier

https://www.datasciencecentral.com/profiles/blogs/roc-curve-explained-in-one-picture

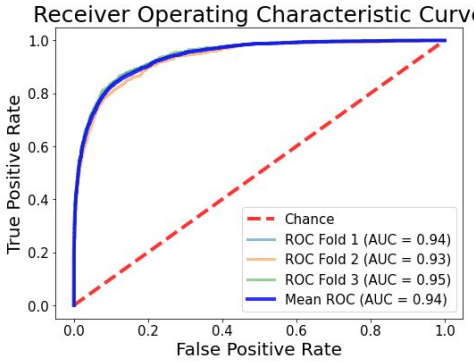
Precision-Recall (PR) curves

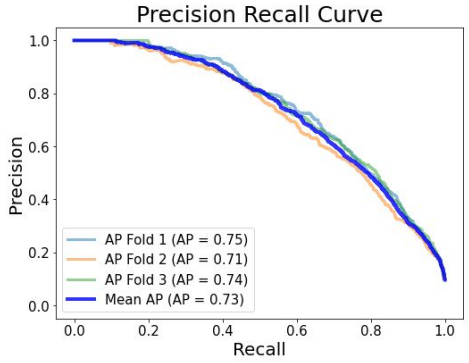
$$TP\_rate = \frac{TP}{P_C}; \quad FP\_rate = \frac{FP}{N_C}.$$

$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN}$$

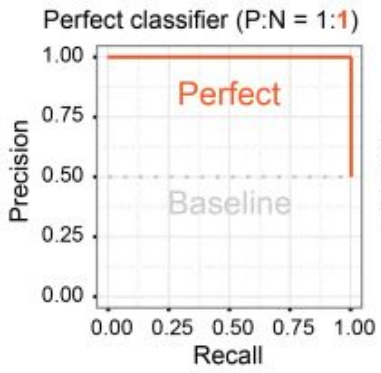
		True class: p	True class: n
Hypothesis output	Y	TP (True Positives)	FP (False Positives)
	N	FN (False Negatives)	TN (True Negatives)
Column counts:		P_C	N_C

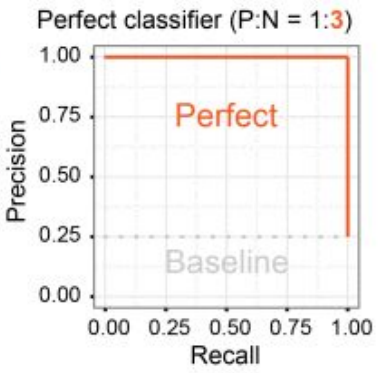
- In highly skewed data sets, the ROC curve may provide an overly optimistic view of an algorithm's performance
  - Assume Pc < Nc : In this case, if a classifier's performance has a large change in the number of false positives, it will not significantly change the FP_rate since the denominator (Nc) is very large
- Under such situations, the **Precision-Recall (PR) curves** can provide a more informative representation of performance assessment
- PR curves exhibit a strong correspondence to ROC curves: A curve dominates in ROC space if and only if it dominates in PR space
- But it considers the ratio of TP with respect to TP+FP, correctly capturing the classifier's performance when the number of **false positives** drastically change





https://www.kaggle.com/c/siim-isic-melanoma-classification/discussion/172453





https://classeval.wordpress.com/introduction/introduction-to-the-precision-recall-plot

Overview

- Machine learning algorithms:
  - Decision Tree
  - Naïve Bayes
  - kNN
  - SVM
  - Linear/logistic Regression
- Ensemble learning: bagging, boosting
- Bias-variance tradeoffs
- Model selection and regularization
  - Feature selection, shrinkage, dimension reduction
- Model validation: cross-validation, metrics
- **Practical issues: Hyperparameter tuning, Feature scaling, data augmentation, imbalanced dataset, high feature dimensions**

Model Evaluation: Final Model?

- How to come up with the final model?
  - Q: If k-fold was used, we have k different models, which one to use?
- Cross-validation is to evaluate a given model!
- Simply use all the data to tune the parameters of the final model

https://machinelearningmastery.com/train-final-machine-learning-model/

Hyperparameter Tuning

- Grid Search
  - Evaluate all the possible combinations of hyperparameter values, using cross-validation (e.g., GridSearchCV() in scikit-learn)
  - Need to specify grid points to search
  - But when you have no idea what value a hyperparameter should have, a simple approach is to try out consecutive powers of 10 (or a smaller number if you want a more fine-grained search)
- Randomized Search
  - When the hyperparameter *search space* is large, it is often preferable to use randomized search
- Holistic approach (AutoML): parameter tuning & model selection (e.g., Auto-Weka, auto-sklearn)
  - See details: https://arxiv.org/pdf/1907.00909.pdf

Data/Feature Scaling

- Different normalization methods:
  - **Z Normalization (Standardization)**:

$$\hat{X}[:,i] = \frac{X[:,i]-\mu_i}{\sigma_i}, (\mu_i = \frac{1}{N} * \sum_{k=1}^{N} X[k,i], \sigma_i = \sqrt{\frac{1}{N-1} * \sum_{k=1}^{N} (X[k,i]-\mu_i)^2}\ )$$

  - **Min-Max Normalization:**

$$\hat{X}[:,i] = \frac{X[:,i]-\min(X[:,i])}{\max(X[:,i])-\min(X[:,i])}$$

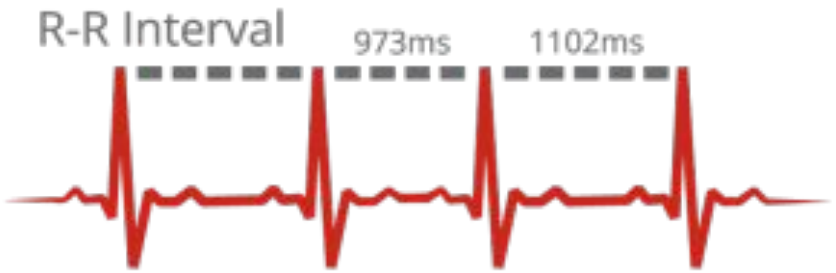
  - **Pure mean subtraction (per person or population)**

- Feature normalization: per person vs. overall population??
  - If personal characteristics are clear, personalized normalization is recommended (e.g., heart rates)

Shanker M, Hu MY, Hung MS, Effect of Data Standardization on Neural Network Training, 1996
https://towardsdatascience.com/understand-data-normalization-in-machine-learning-8ff3062101f0

Data/Feature Scaling

- RR interval features were directly related with the heart rate



- Heart rate heterogeneity, possibly due to measurement errors and individual differences, could lower the classification accuracy
- Normalizing features based on the mean value of all RR intervals (per person, per ECG recordings)

Heartbeat Classification Using Normalized RR Intervals and Morphological Features, 2014
https://www.hindawi.com/journals/mpe/2014/712474/

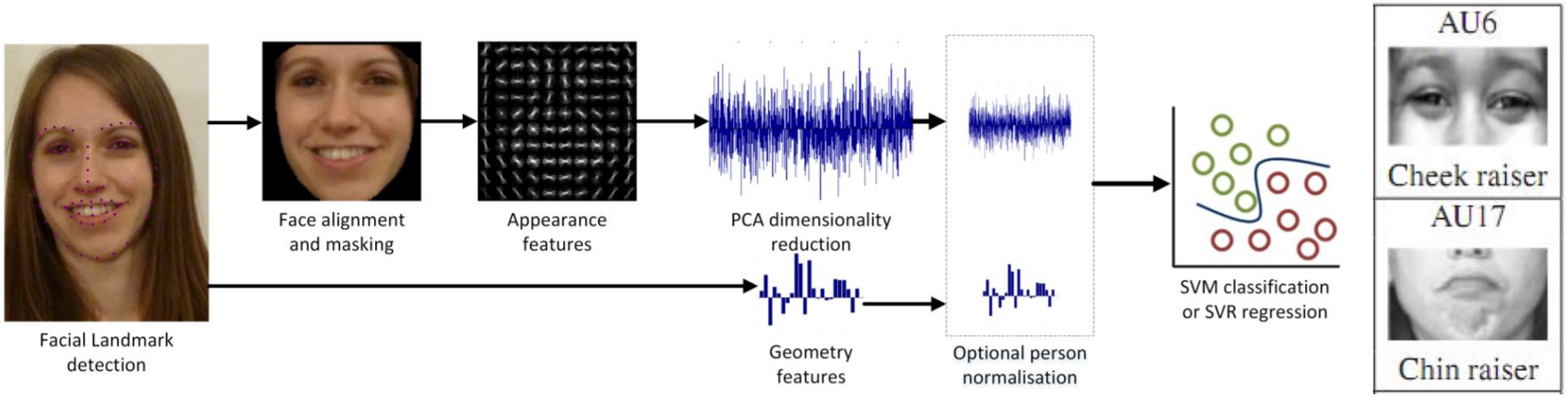
Data/Feature Scaling

- Values of an atmospheric pressure sensor have a large variance due to different weather conditions and activities
- There are significant influences on different weather conditions between various measurement days and short-term changes as they occur during e.g. stair climbing
- To compensate this effect, for a given time window of interest, it's possible to subtract the mean value to normalize pressure values, and then, a standard z-normalization can be applied

CNN-based sensor fusion techniques for multimodal human activity recognition, ISWC 2017

Data/Feature Scaling

- **Automatic detection of facial Action Units (AUs)** – binary classification
  - Issue: neutral facial expression may not be known; it may be significantly different across people (e.g., smiley vs. frowny persons)
  - Normalization by subtracting the median value of face descriptors of a given person



	SEMAINE					DISFA							BP4D		
	AU2	AU12	AU17	AU25	AU45	AU2	AU5	AU6	AU9	AU15	AU17	AU20	AU4	AU6	Mean
Default	0.34	0.58	0.15	0.46	0.38	0.13	0.14	0.44	0.23	0.28	0.23	0.10	0.44	0.76	0.33
Normalization	**0.59**	**0.61**	**0.44**	**0.52**	**0.42**	**0.26**	**0.17**	**0.58**	**0.46**	**0.48**	**0.43**	**0.22**	**0.53**	**0.79**	**0.46**

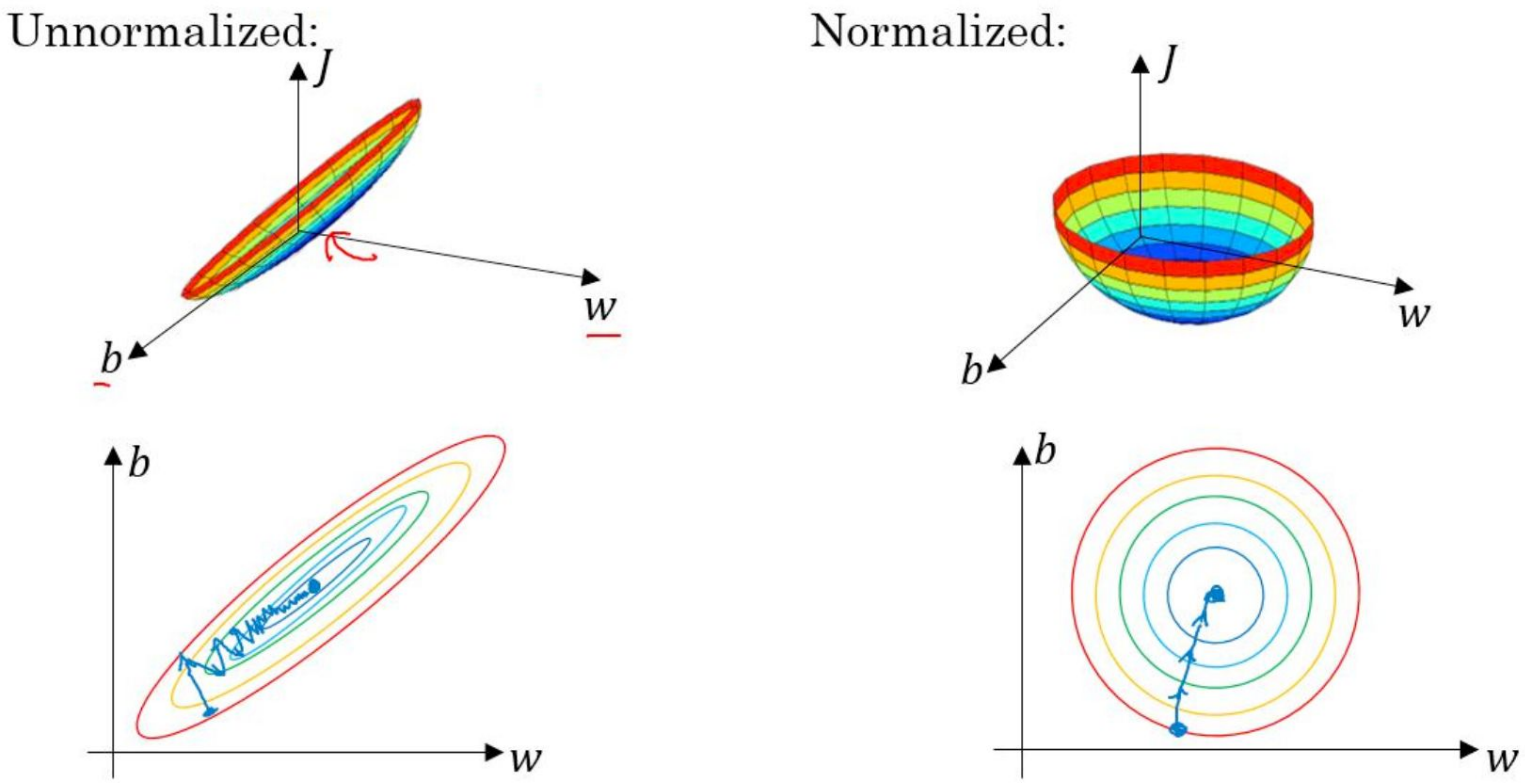
Cross-dataset learning and person-specific normalisation for automatic Action Unit detection, Tadas Baltrusaitis and Marwa Mahmoud and Peter Robinson, 2015 https://www.cl.cam.ac.uk/~pr10/publications/fera15.pdf

Data/Feature Scaling

- Effects on learning algorithms:
  - Regression: almost no effect
  - Dimension reduction (PCA): normalization is a must
  - Distance-based algorithms (kNN): normalization gives equal weights for each feature (completely changes its meaning)
  - Tree-based algorithms: almost no effect (decision boundaries can be correctly made without scaling)
  - Neural networks: stabilize gradient descent (and may speed up); e.g., batch normalization of hidden nodes

- Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift Sergey Ioffe, Christian Szegedy, 2015
- Understand Data Normalization in Machine Learning https://towardsdatascience.com/understand-data-normalization-in-machine-learning-8ff3062101f0

Data/Feature Scaling



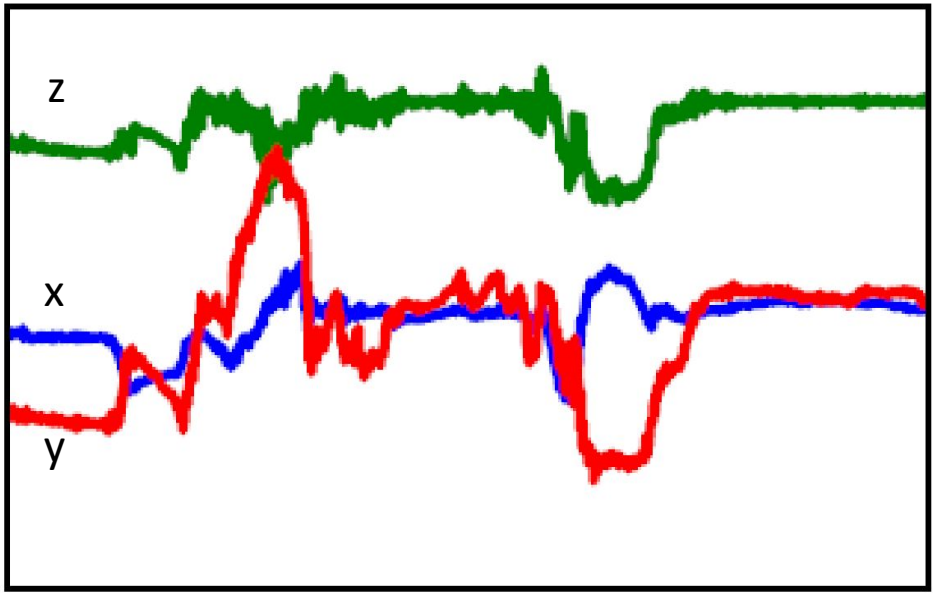
https://www.youtube.com/watch?v=FDCfw-YqWTE&feature=youtu.be&ab_channel=Deeplearning.ai

Data Augmentation

- Increasing the size of a training dataset
- Image transforms: shift, shear, scaling, rotating

Data Augmentation

- Data augmentation of accelerometer signals



Original data

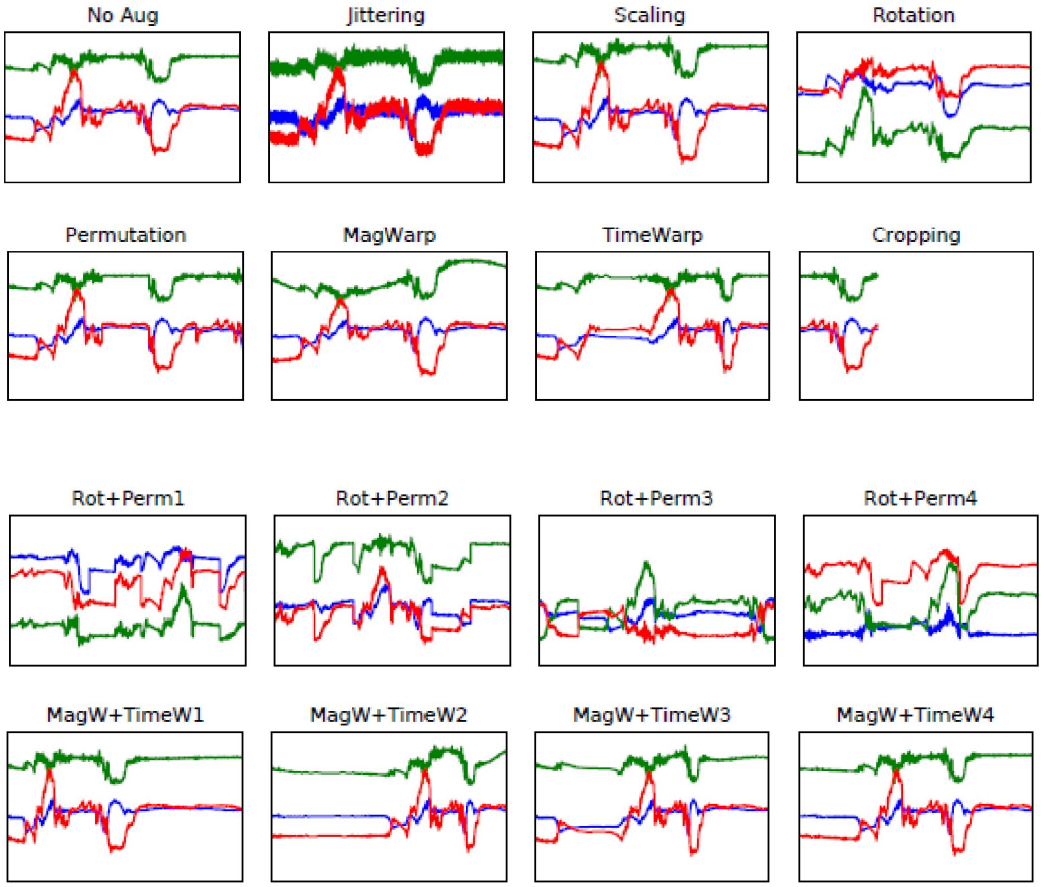
- **Data Augmentation** of Wearable Sensor Data for Parkinson's Disease Monitoring using Convolutional Neural Networks, ICMI 2017
- Understanding data augmentation for classification: when to warp? DICTA 2016 https://arxiv.org/pdf/1609.08764.pdf

Data Augmentation

- Data augmentation of accelerometer signals
  - Arbitrary rotations (Rot) to the existing data
  - Temporal perturbation
    - Permutation (Perm): randomly perturb the temporal location of within-window events
    - Time-warping (TimeW): smoothly distorting the time intervals between samples, the temporal locations of the samples can be changed using time-warping
  - Scaling (Scale): changing the magnitude of the data in a window by multiplying by a random scalar
  - Magnitude-warping (MagW): changing the magnitude of each sample by convolving the data window with a smooth curve
  - Jittering (Jitter): simulating additive sensor noise to increase robustness against multiplicative and additive noise
  - Cropping (Crop): similar to image cropping or window slicing; used to lower the dependency on event locations

Data Augmentation of Wearable Sensor Data for Parkinson's Disease Monitoring using Convolutional Neural Networks, ICMI 2017

Data Augmentation



Data Augmentation of Wearable Sensor Data for Parkinson's Disease Monitoring using Convolutional Neural Networks, ICMI 2017

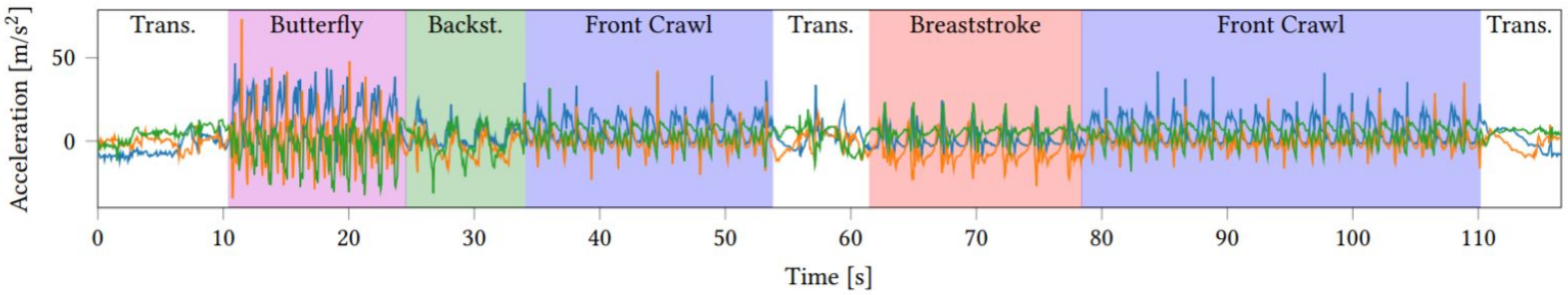
Data Augmentation

Table 1: The results of PD motor state classification with various data augmentation methods. R,P,T,M represent *Rot*, *Perm*, *TimeW*, *MagW*, respectively.

	SVM	CNN	Jitter	Scale	Crop	Rot	Perm
Train	98.82	99.92	99.78	99.84	65.77	100.0	99.33
Test	70.72	77.54	77.52	79.46	73.58	**82.62**	81.16
	MagW	TimeW	P,T	R,P	R,T	R,P,T	R,P,T,M
Train	100.0	94.67	96.63	99.08	94.70	94.43	94.20
Test	79.33	82.00	81.75	**86.76**	85.01	**86.88**	85.60

Data Augmentation of Wearable Sensor Data for Parkinson's Disease Monitoring using Convolutional Neural Networks, ICMI 2017

Data Augmentation

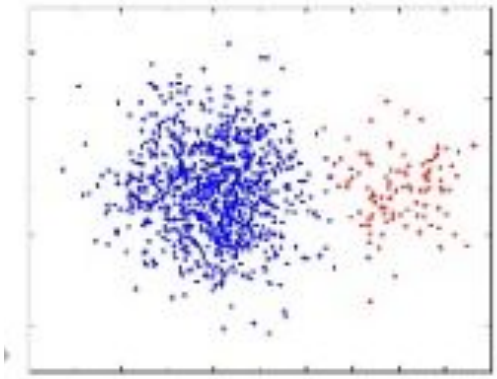


- **Time-scaling**. to simulate a swimmer swimming faster or slower than in the original recording.
- **Noise.** In order to increase the robustness of our methods with respect to random fluctuations we add zero mean Gaussian noise to each normalized input window
- **Reversing.** During training we transform the sensor axes with a certain probability in order to simulate how the signals would look like if the watch was worn on the opposite wrist.
- **Rotation.** Apply random rotations around the x-axis (parallel to the arm) in order to simulate the watch being rotated around the wrist.

Swimming style recognition and lap counting using a smartwatch and deep learning, ISWC 2019

Imbalanced Dataset

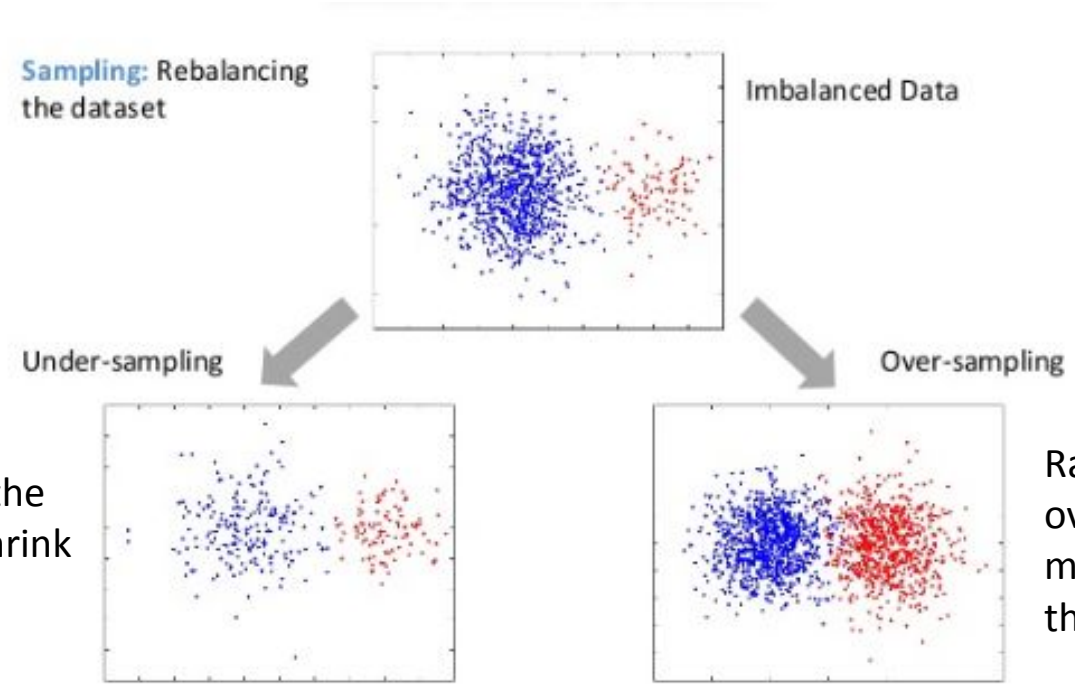
- Imbalanced dataset: Yes: 1,000,000 vs. No: 1,000
  - The minority class may be outnumbered, but not necessarily rare
  - Also there are datasets with high dimensionality and small sample size (risk of overfitting)



Learning from Imbalanced Data, TKDD 2009

Imbalanced Dataset

- Imbalanced dataset: Yes: 1,000,000 vs. No: 1,000
  - Rebalancing w/ sampling



under-sampling the majority class (shrink the majority)

Random over-sampling the minority class (expand the minority)

https://www.datasciencecentral.com/profiles/blogs/handling-imbalanced-data-sets-in-supervised-learning-using-family

Imbalanced Dataset: Less Learning?

- Problems with imbalanced dataset in training decision trees
  - Successive partitioning of the dataspace results in fewer and fewer observations of minority class samples resulting in fewer leaves describing minority concepts and successively weaker confidences estimates
  - Concepts that have dependencies on different feature space conjunctions can go unlearned by the sparseness introduced through partitioning (curse of dimensionality)
- As a result, learning from imbalanced dataset has a detrimental effect

Learning from Imbalanced Data, TKDD 2009

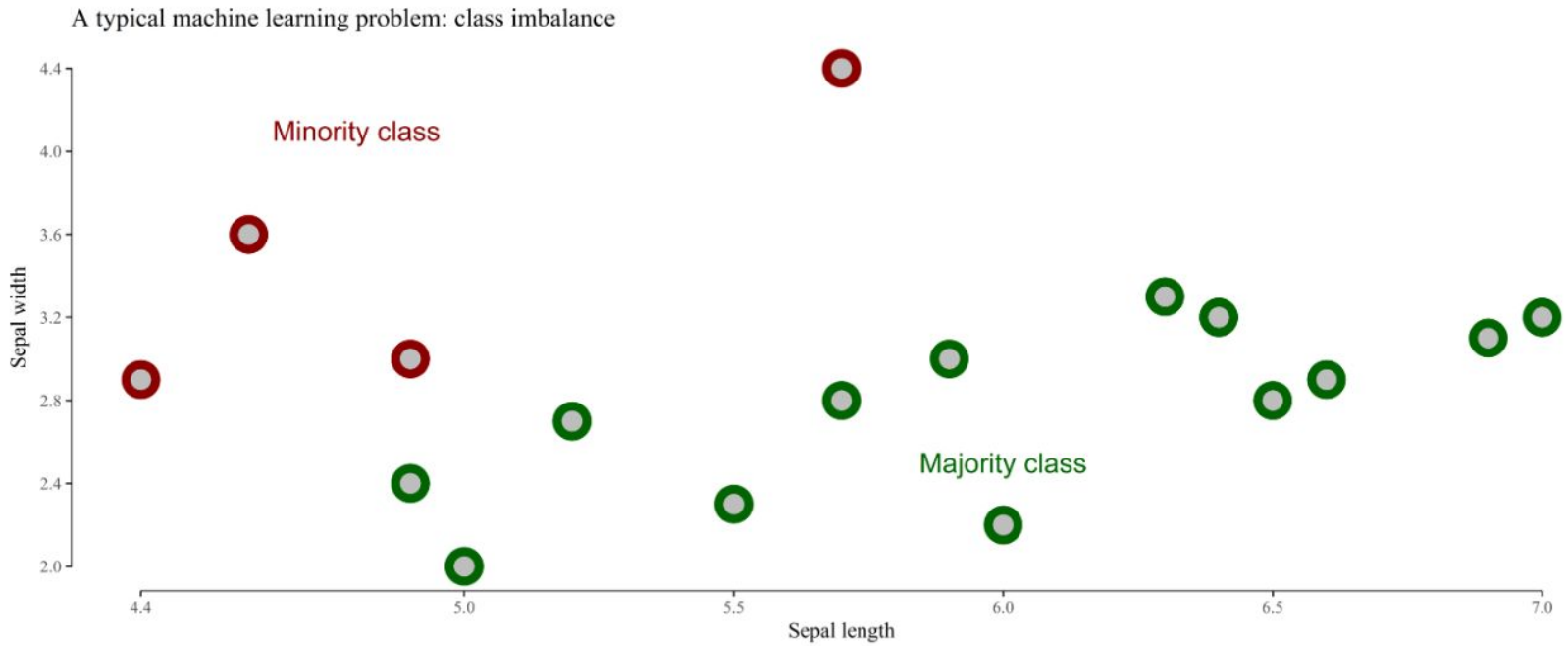
Imbalanced Dataset

- Common approach: modifying the training data (the class distribution)
  - Key idea: if dataset is imbalanced, modify data distribution and generate a balanced dataset (random sampling!)
  - Random under-sampling the majority class (shrink the majority)
    - Concern: Loss of important concepts
    - Informed under sampling: e.g., selecting representative subset
  - Random over-sampling the minority class (expand the minority)
    - Concern: Overfitting due to multiple "tied" instances
  - **Synthetic sampling with data generation**: e.g., synthetic minority oversampling techniques (SMOTE) – use k-NN to select x's k minority samples to synthesize a new sample
- Other methods: cost-sensitive methods, kernel-based methods, and active learning methods

Learning from Imbalanced Data, TKDD 2009

Imbalanced Dataset

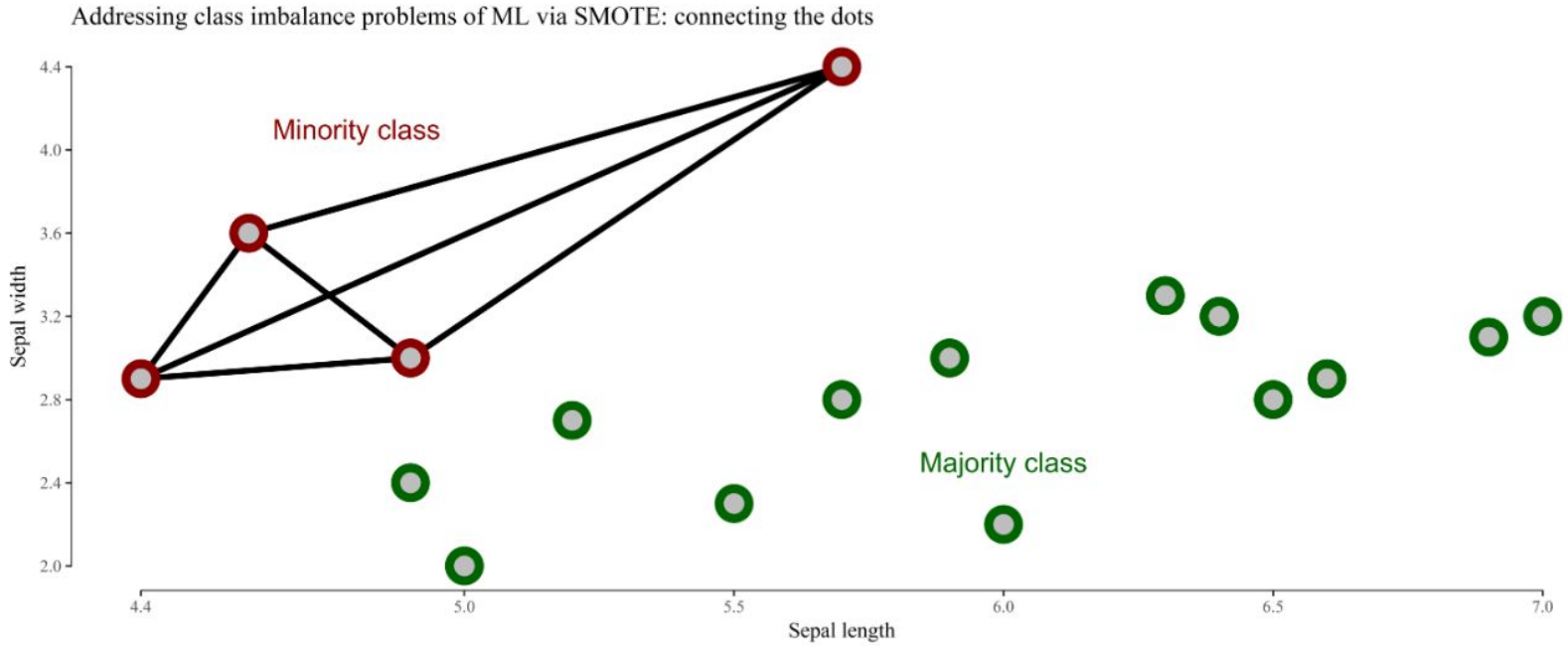
- SMOTE: Synthetic minority oversampling
  - Synthesize new minority instances between existing (real) minority instances



http://rikunert.com/SMOTE_explained

Imbalanced Dataset

- SMOTE: Synthetic minority oversampling

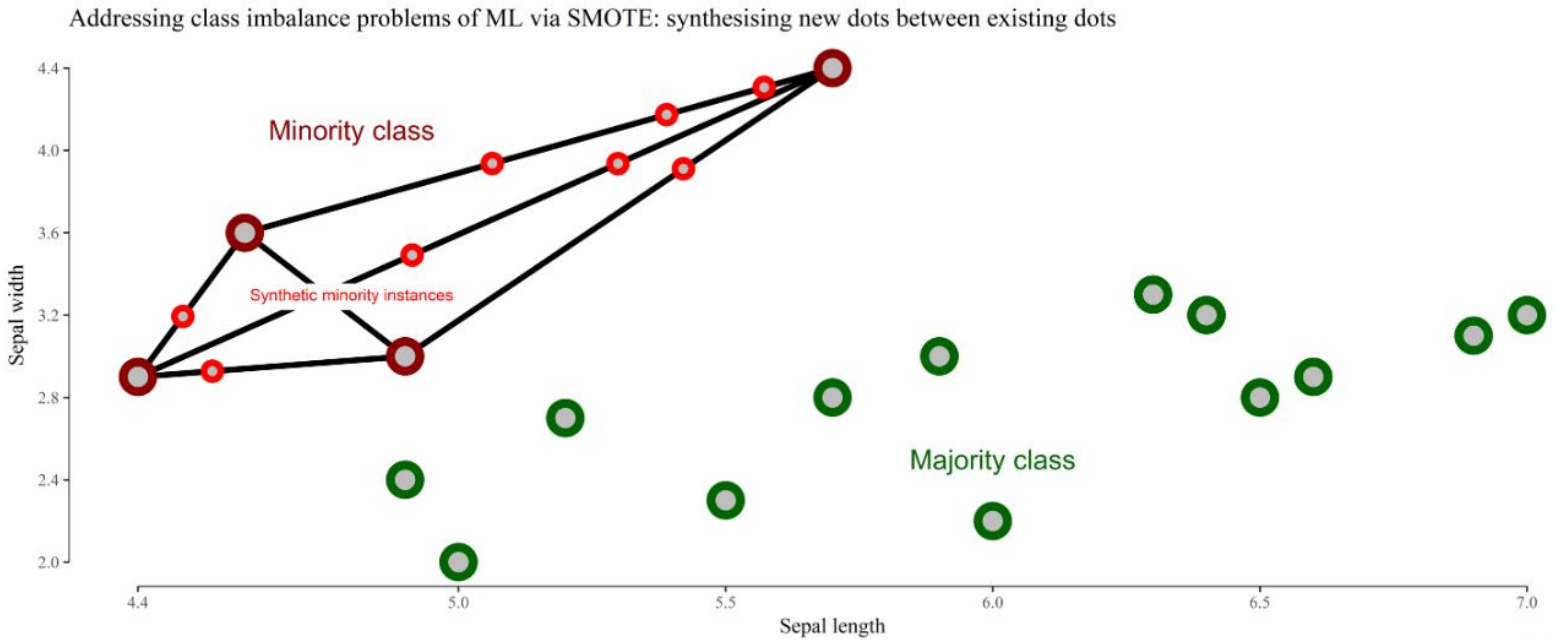


Imagine that SMOTE draws lines between existing minority instances like this

http://rikunert.com/SMOTE_explained

Imbalanced Dataset

- SMOTE: Synthetic minority oversampling

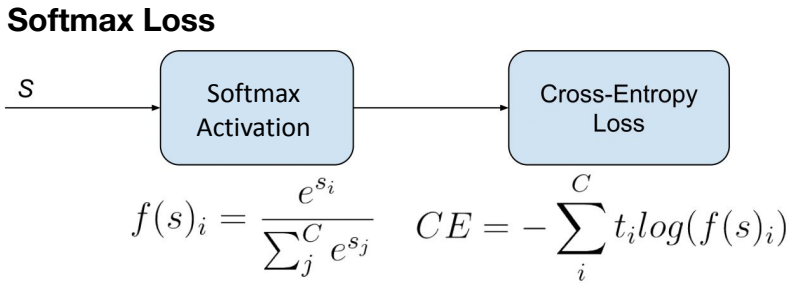


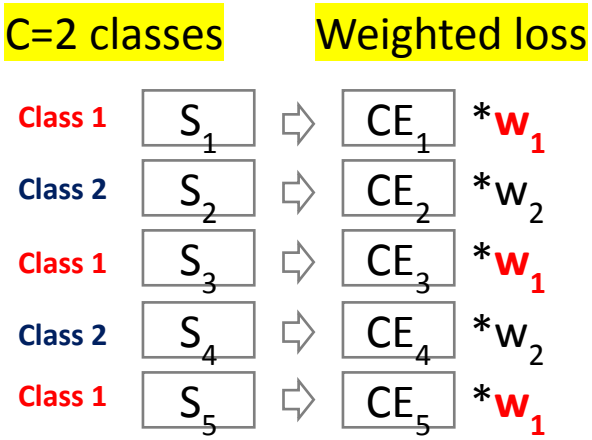
SMOTE then imagines new, synthetic minority instances somewhere on these lines

http://rikunert.com/SMOTE_explained

Imbalanced Dataset

- Cost-sensitive learning: e.g., weight balancing
  - Telling the model to "pay more attention" to samples from an under-represented class
  - Weighting (or scaling) the loss function (during training only); e.g., **class_weight** parameter in sklearn

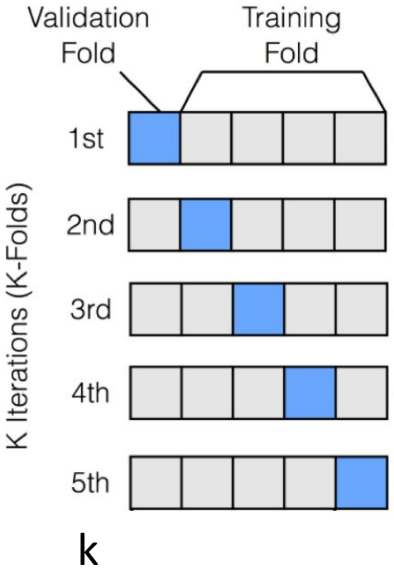




https://gombru.github.io/2018/05/23/cross_entropy_loss/
https://www.tensorflow.org/tutorials/structured_data/imbalanced_data

Model Evaluation w/ Rebalancing

- Correct way to do cross-validation w/ rebalancing:
  1. Divide the samples into K cross-validation folds (groups) at random
  2. For each fold k = 1, 2, . . . , K
     1. **Rebalancing** the dataset w/ resampling techniques (e.g., SMOTE) using all of the samples **except those in fold k**
     2. Find a subset of "good" predictors that show fairly strong (univariate) correlation with the class labels, using all of the samples except those in fold k
     3. Using just this subset of predictors, build a multivariate classifier, using all of the samples **except those in fold k**
     4. Use the classifier to predict the class labels for the samples in fold k



Influence of Data Imbalance on Performance Measures

- Existing measures are all vulnerable to class imbalance: precision, recall, **f1-score**
- Because class distribution greatly influences performance measures
- How can we compare the performance of different algorithms? (metrics should NOT be influenced by the class imbalance)
  - ~~Approach 1: **Rebalance & Compare f1-scores**~~
    - Holdout data have the same imbalance issue; existing measure will suffer from imbalance!!
  - Approach 2: Use **AUC-ROC** or AUC-PR curves

Learning from Imbalanced Data, TKDD 2009

High Feature Dimensions

- High dimensional setting: # of features could be much larger than # of samples
- If we select a subset of features, it is likely that we just identified simply **one of many possible models**
- Be careful about reporting errors; for example, with multiple regression, don't report R^2 or RSS values (these tend to be close to be near perfect values: 1 and 0, respectively)
- Use cross-validations (w/ proper holdouts, including subset selection)

Summary

- Machine learning algorithms:
  - Decision Tree
  - kNN
  - Linear/logistic Regression
- Ensemble learning: bagging, boosting
- Model selection and regularization
  - Feature selection, shrinkage, dimension reduction
- Model validation: cross-validation, metrics
- Practical issues: Hyperparameter tuning, feature scaling, data augmentation, imbalanced dataset, high feature dimensions