UCLA Samueli Computer Science

CS M146 Discussion: Week 8

Ensemble Method, Multi-Class Classification, ML Evaluation

Junheng Hao

Friday, 02/26/2021

UCLA
Engineer Change.

Roadmap

- Announcement
- Ensemble Method
- Multi-Class Classification
- ML Evaluation

UCLA
Engineer Change.

Announcements

- **5:00 pm PST, Feb 26 (Friday):** Weekly Quiz 8 released on Gradescope.
- **11:59 pm PST, Feb 28 (Sunday):** Weekly Quiz 8 closed on Gradescope!
  - Start the quiz before **11:00 pm PST, Feb 28** to have the full 60-minute time
- **Problem set 3** released on CCLE, submission on Gradescope.
  - Please assign pages of your submission with corresponding problem set outline items on GradeScope.
  - You need to submit code and the results required by the problem set
  - Due on **Today 11:59pm PST, Feb 26 (Friday)!**

Late Submission of PS will NOT be accepted!

UCLA
Engineer Change.

About Quiz 8

- Quiz release date and time: **Feb 26, 2021 (Friday) 05:00 PM PST**
- Quiz due/close date and time: **Feb 28, 2021 (Sunday) 11:59 PM PST**
- You will have up to **60 minutes** to take this exam. → Start before **11:00 PM** Sunday
- You can find the exam entry named "Week 8 Quiz" on GradeScope.
- Topics: **Ensemble Method, Multi-Class Classification, ML Evaluations**
- Question Types
  - True/false, multiple choices
  - Some questions may include several subquestions.
- Some light calculations are expected. Some scratch paper and one scientific calculator (physical or online) are recommended for preparation.

Quiz 7 Review: Kernel SVM

Q6 Kernel SVM

2 Points

We can introduce non-linearity to SVM using the kernel trick. Instead of searching for a hyperplane $\mathbf{w}^T\mathbf{x} + b$ that maximizes the margin, we are looking for $\mathbf{w}^T\phi(\mathbf{x}) + b$ where ϕ is the non-linear basis function.

Which **one** of the following statements is **wrong** about kernel SVM?

- (●) We can learn the optimal value of the weights $\mathbf{w}$ using only the kernel function.
- ( ) We can predict the label of a new sample using only the kernel function.
- ( ) If we apply kernel functions, non-separable data may be separable.
- ( ) A valid kernel function should have a positive-semidefinite kernel matrix.

	Linear SVM $\boldsymbol{w}^\top \boldsymbol{x} + b$	Kernel SVM $\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) + b$
Weight parameter	$\boldsymbol{w} = \sum_n \alpha_n y_n \boldsymbol{x}_n$	$\boldsymbol{w} = \sum_n \alpha_n y_n \boldsymbol{\phi}(\boldsymbol{x}_n)$
Predicting new data	$\mathrm{SIGN}\left(\sum_n y_n \alpha_n \left(\boldsymbol{x}_n^T \boldsymbol{x}\right) + b\right)$	$\mathrm{SIGN}(\sum_n y_n \alpha_n k(\boldsymbol{x}_n, \boldsymbol{x}) + b)$

UCLA
Engineer Change.

Quiz 7 Review: Kernel SVM

Q4 SVM on non-separable data

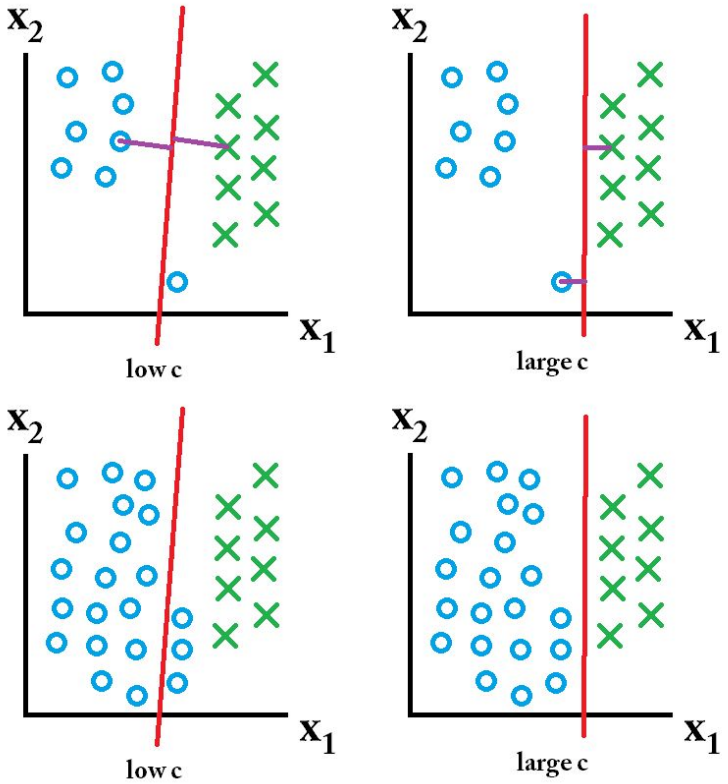
2 Points

Which **one** of the following statements is **wrong** about SVM applied to non-separable data (soft-margin SVM)?

- ○ The use of slack variables permits samples to be misclassified.
- ○ As we increase the value of the hyperparameter C, the margin of the learned hyperplane decreases.
- ○ As we decrease the value of the hyperparameter C, the training accuracy of the learned SVM may decrease.
- ⊙ During training, we do not only minimize $\frac{1}{2}\|\omega\|_2^2$, but also minimize $\sum(1-\xi_n)$ so that the value of slack variable can be controlled.

SVM: Understanding C

- The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example.
- For large values of C, the optimization will choose **a smaller-margin hyperplane** if that hyperplane does a better job of getting all the training points classified correctly.
- Conversely, a very small value of C will cause the optimizer to look for **a larger-margin separating hyperplane**, even if that hyperplane misclassified more points.

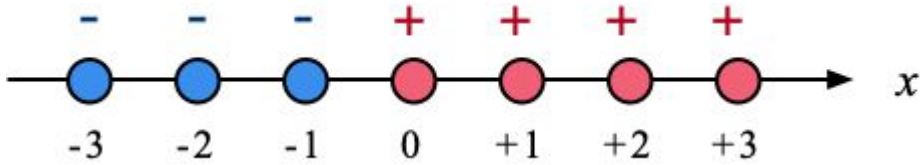


Reference: https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel

Additional: SVM on 1-dim data

$$\min_{w,b} \quad \frac{1}{2} w^T w + \sum_{i=1}^{m} \epsilon_i$$

$$\text{subject to: } y_i \left(w^T x_i + b \right) \geq 1 - \epsilon_i, \epsilon_i \geq 0, \forall i$$



- What if C is super large, i.e. C→ +inf?
- What is C is super small, i.e. C→ 0 (a very small positive number, such as 0.0001)?

Additional: SVM on 1-dim data (code)

Colab link:
https://colab.research.google.com/drive/1Ru_gN8UikD_fGY3DfarHTfXQQDg4b-D4?usp=sharing

```
[1] from sklearn import svm
    import numpy as np
```

```
[2] X = [[-3], [-2], [-1], [0], [1], [2], [3]]
    y = [0, 0, 0, 1, 1, 1, 1]
    X, y = np.array(X), np.array(y)
```

```
[3] def print_svm_decision(X, y, C):
      clf = svm.LinearSVC(C=C)
      clf.fit(X,y)
      decision_function = clf.decision_function(X)
      support_vector_indices = np.where( (2*y-1)*decision_function <= 1)[0]
      print("Support Vectors are: ", support_vector_indices)
      print("Predicting y=wx+b:", decision_function)
```

```
print_svm_decision(X, y, C=1)

Support Vectors are:  [2 3]
Predicting y=wx+b: [-2.36363017 -1.45454339 -0.54545661  0.36363017  1.27271694  2.18180372
  3.0908905 ]
```

```
[5] print_svm_decision(X, y, C=0.5)

    Support Vectors are:  [2 3 4]
    Predicting y=wx+b: [-1.75000807 -1.08333882 -0.41666956  0.24999969  0.91666894  1.5833382
      2.25000745]
```

```
[6] print_svm_decision(X, y, C=0.1)

    Support Vectors are:  [1 2 3 4 5]
    Predicting y=wx+b: [-1.09999783 -0.69999856 -0.29999929  0.09999998  0.49999925  0.89999852
      1.29999779]
```

Case: C is suffciently small and every vector is within the margin, i.e. every data point is a support vector.

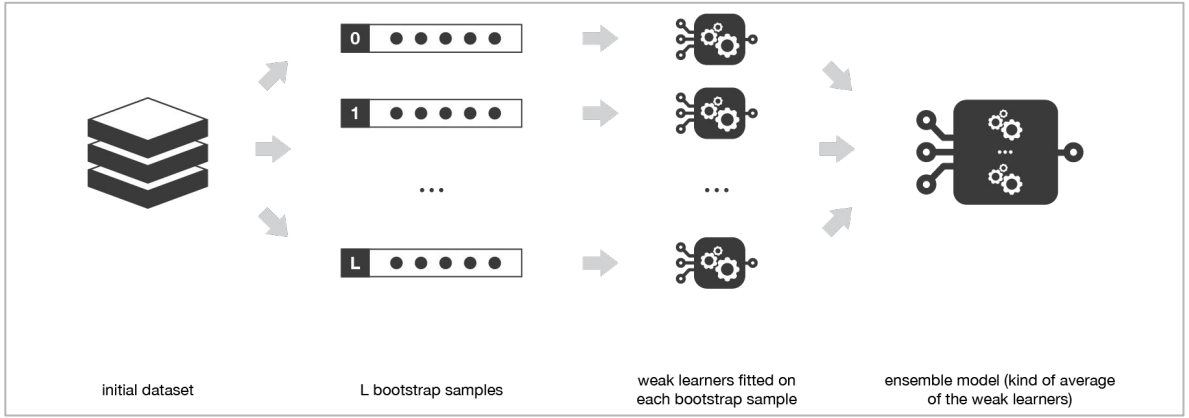
```
[7] print_svm_decision(X, y, C=0.01)

    Support Vectors are:  [0 1 2 3 4 5 6]
    Predicting y=wx+b: [-0.44399465 -0.29014848 -0.13630231  0.01754386  0.17139003  0.3252362
      0.47908237]
```

Ensemble: Bagging and Boosting

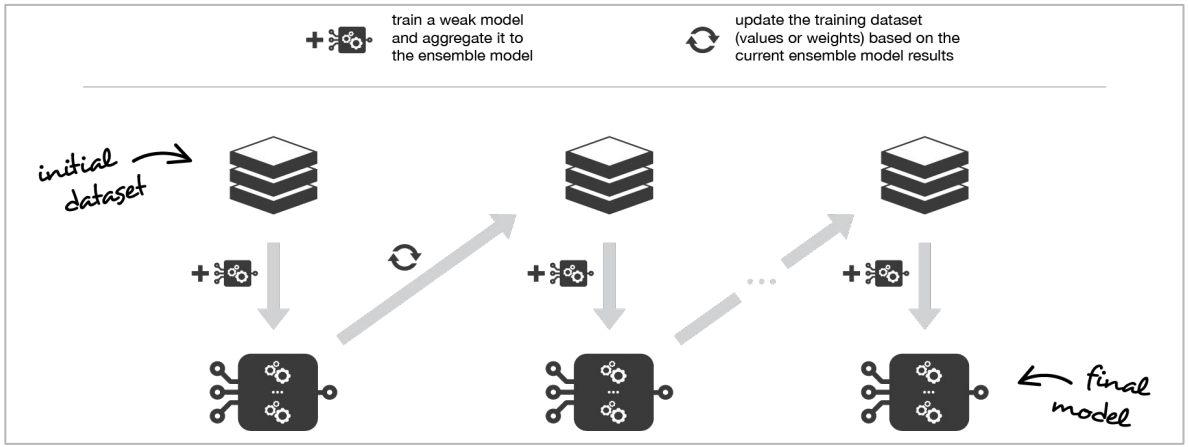
Bagging

"Parallel Learner"



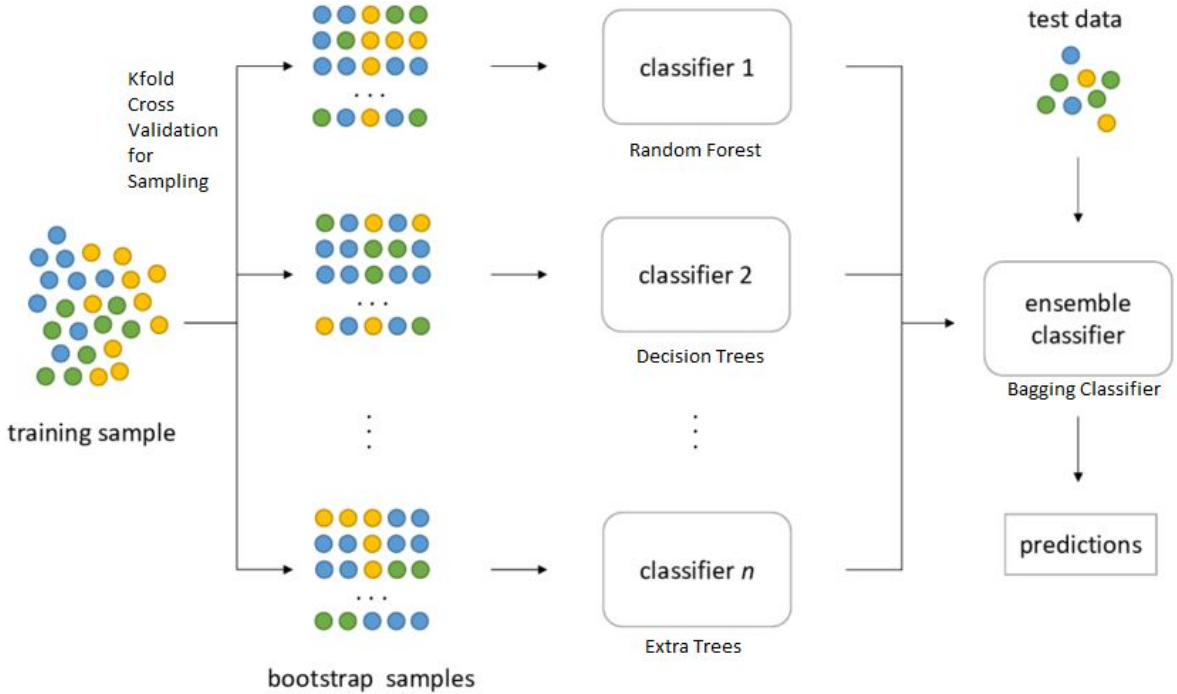
Boosting

"Sequential Learner"



Ensemble: Bagging (Bootstrap Aggregation)

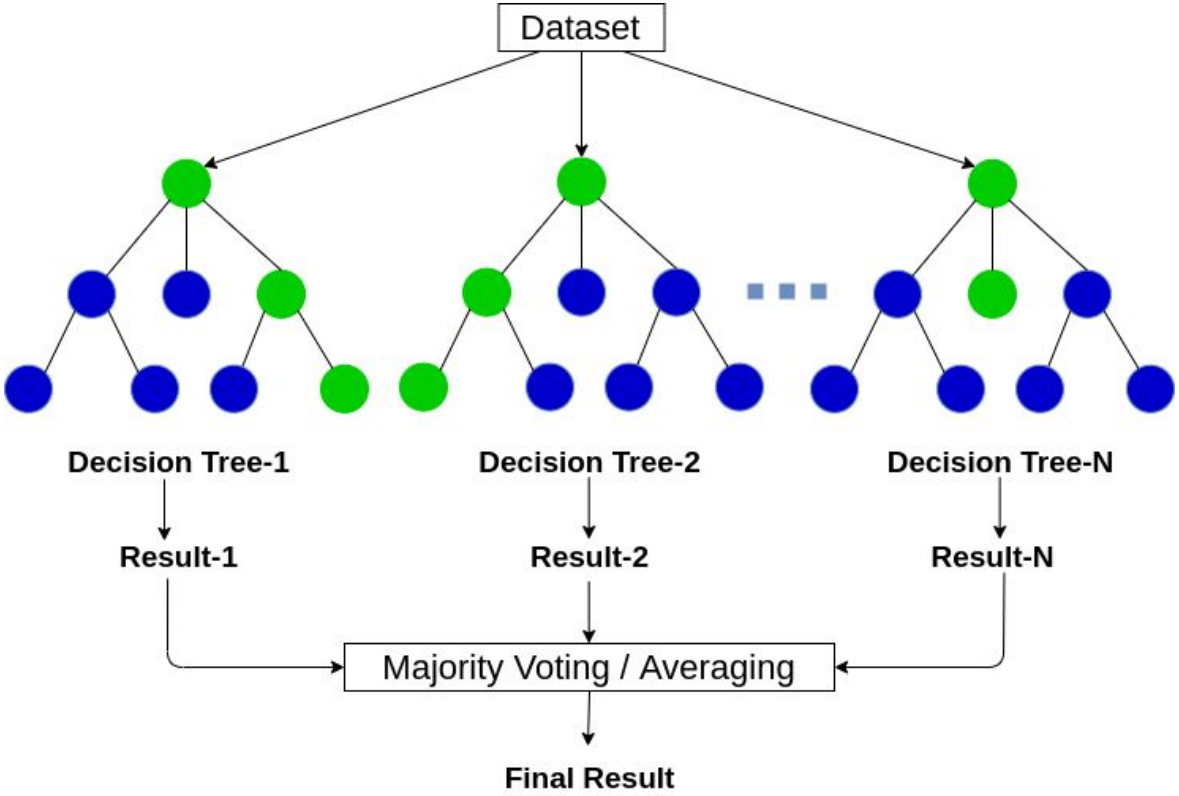
- **"Multiple"** dataset and **multiple** classifier



Bagging Classifier Process Flow

Ensemble: Bagging (Random Forest)

- **Single:** Decision Tree → **Bagging:** Random Forest



Ensemble: Boosting (Adaboost)

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing the weighted classification error

$$\epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$

  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
  3. Update weights on training points

$$w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$

  and normalize them such that $\sum_n w_{t+1}(n) = 1$

- Output the final classifier

$$h[\boldsymbol{x}] = \text{sign}\left[\sum_{t=1}^{T} \beta_t h_t(\boldsymbol{x})\right]$$

1. Calculating error

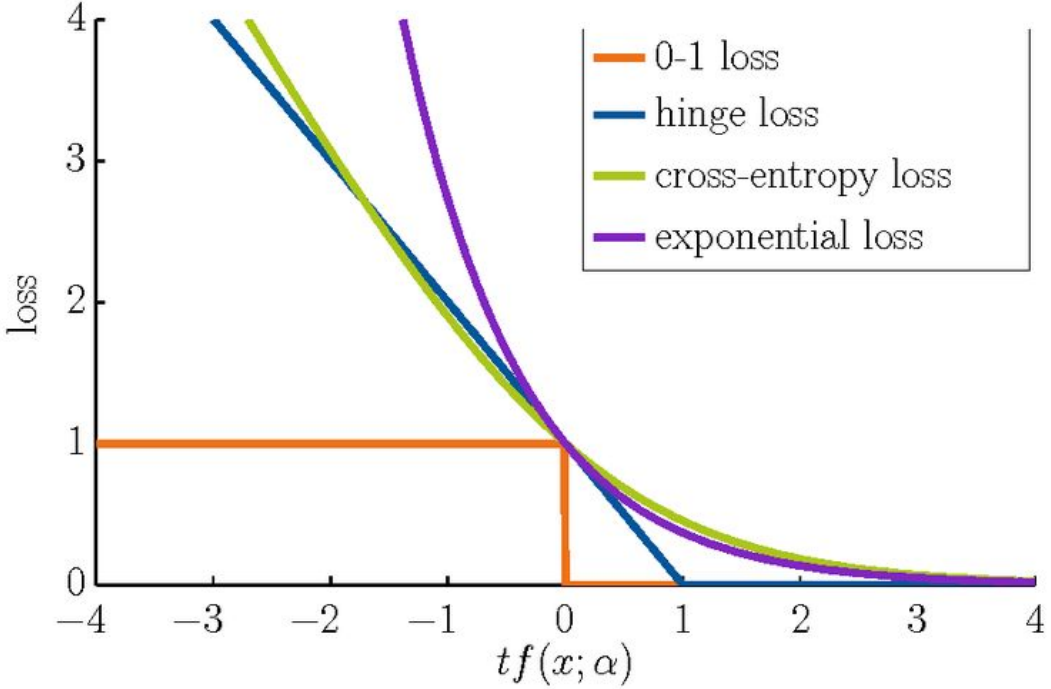
2. Calculating classifier weights of t

3. Reweighting training points

Bagging Classifier

Ensemble: Adaboost Loss Function

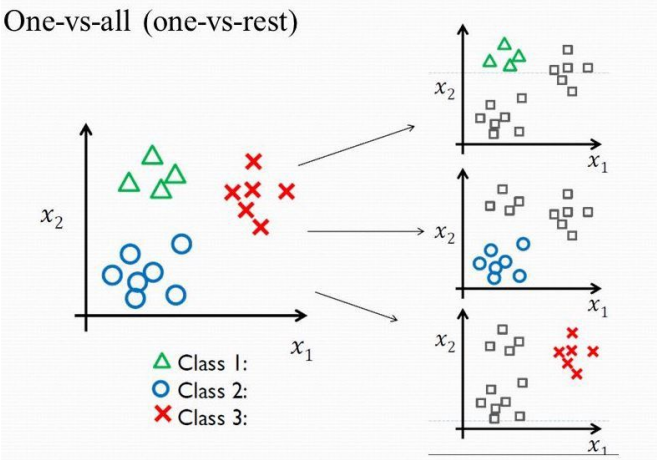
- **Exponential loss**, instead of 0/1 loss



Multi-Class Classification: Two modes

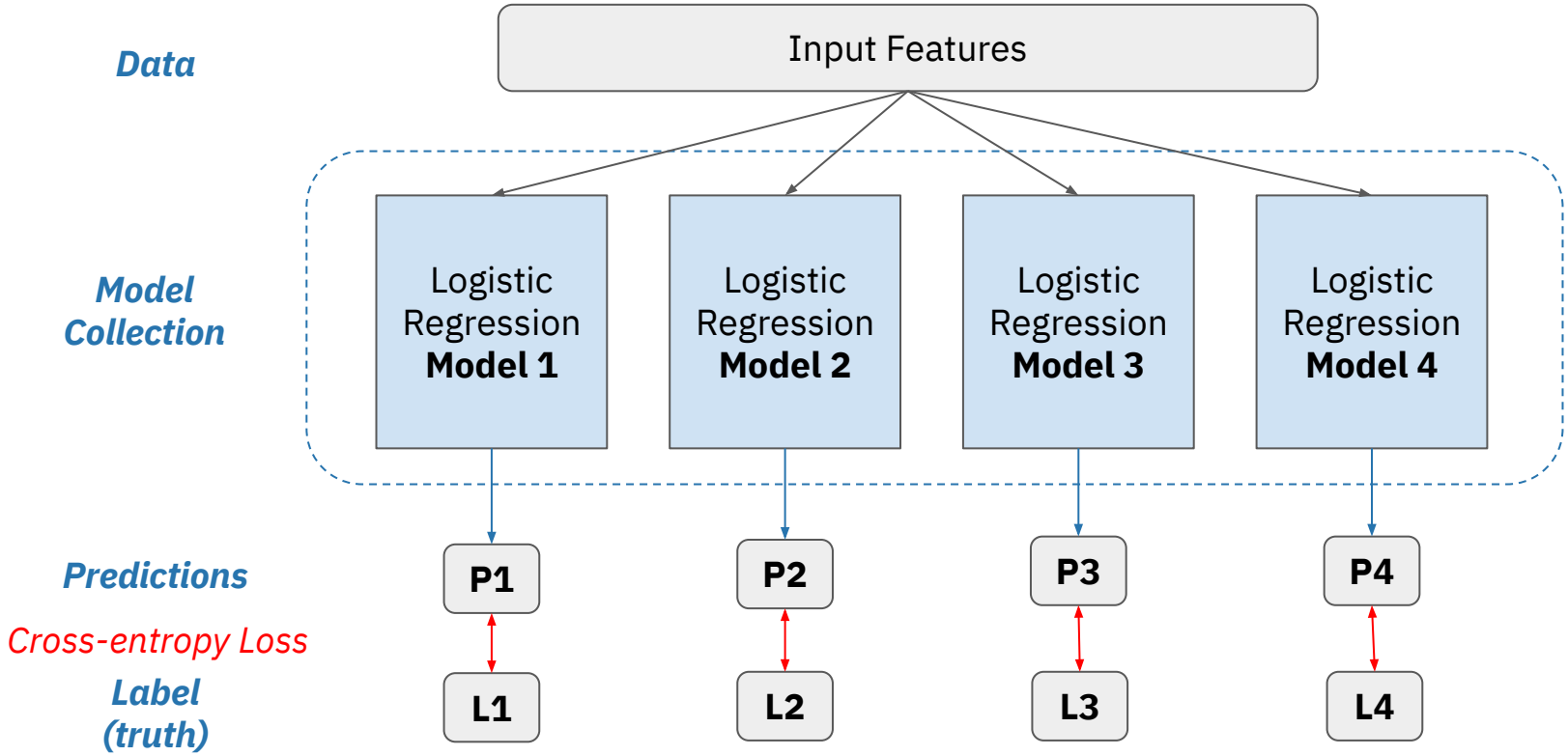
Given **C** classes and **N** data points each class:

Comparison	**One-vs-one**	**One-vs-rest (all)**
# Binary Classifies		
# Training Data		
Pros		
Cons		



UCLA
Engineer Change.

Logistic Regression: Multiclass Case



Multinomial Logistic Regression

Model:

Model

For each class C_k, we have a parameter vector $\boldsymbol{\theta}_k$ and model the probability of class C_k as

$$p(y = C_k|\boldsymbol{x}; \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K) = \frac{e^{\boldsymbol{\theta}_k^{\mathrm{T}}\boldsymbol{x}}}{\sum_{k'} e^{\boldsymbol{\theta}_{k'}^{\mathrm{T}}\boldsymbol{x}}} \quad \leftarrow \quad \text{This is called } \textit{softmax}$$

Decision boundary: assign $\boldsymbol{x}$ with the label that is the maximum of

$$\arg\max_k P(y = C_k|\boldsymbol{x}; \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K) \rightarrow \arg\max_k \boldsymbol{\theta}_k^{\mathrm{T}}\boldsymbol{x}$$

Likelihood

$$\begin{aligned}\sum_n \log P(y_n|\boldsymbol{x}_n; \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K) &= \sum_n \log \prod_{k=1}^{K} P(y = C_k|\boldsymbol{x}_n; \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K)^{y_{nk}} \\ &= \sum_n \sum_k y_{nk} \log P(y = C_k|\boldsymbol{x}_n; \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K)\end{aligned}$$

MLR: Cost Function and Optimization

Likelihood

$$\sum_n \log P(y_n|\boldsymbol{x}_n;\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_K) = \sum_n \log \prod_{k=1}^{K} P(y=C_k|\boldsymbol{x}_n;\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_K)^{y_{nk}}$$
$$= \sum_n \sum_k y_{nk} \log P(y=C_k|\boldsymbol{x}_n;\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_K)$$

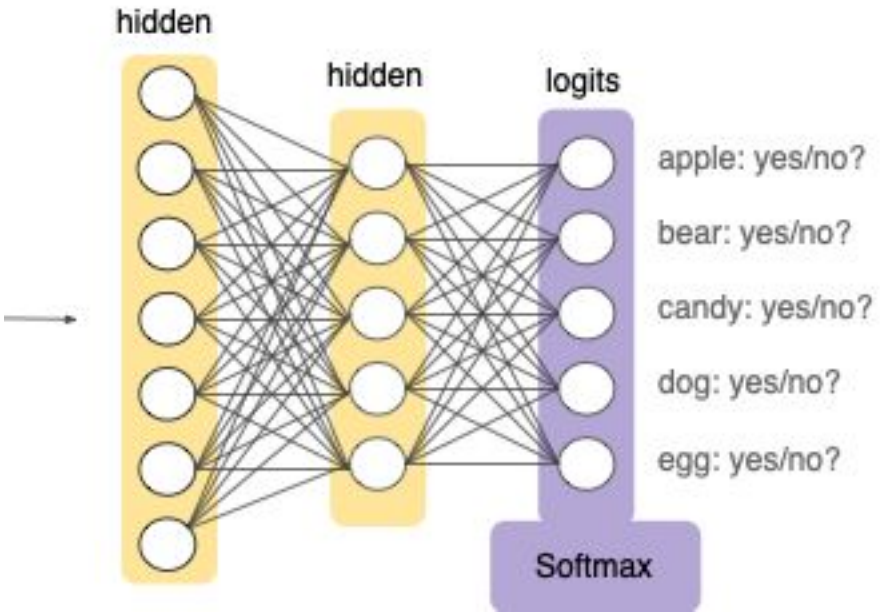
Cost Function

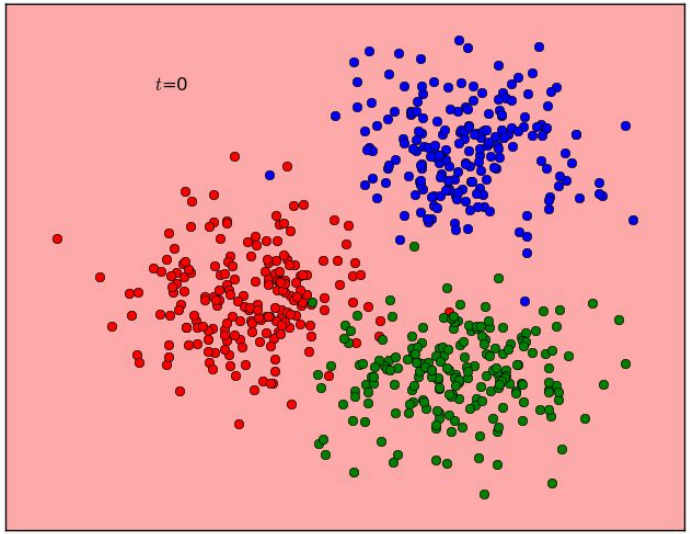
$$J(\boldsymbol{\theta}_1,\boldsymbol{\theta}_2,\ldots,\boldsymbol{\theta}_K) = -\sum_n \sum_k y_{nk} \log P(y=C_k|\boldsymbol{x}_n;\boldsymbol{\theta}_1,\boldsymbol{\theta}_2,\ldots,\boldsymbol{\theta}_K)$$
$$= -\sum_n \sum_k y_{nk} \log \left(\frac{e^{\boldsymbol{\theta}_k^{\mathrm{T}}\boldsymbol{x}}}{\sum_{k'} e^{\boldsymbol{\theta}_{k'}^{\mathrm{T}}\boldsymbol{x}}} \right)$$
$$= -\sum_n \sum_k y_{nk}\boldsymbol{\theta}_k^{\mathrm{T}}\boldsymbol{x} - y_{nk} \log \left(\sum_{k'} e^{\boldsymbol{\theta}_{k'}^{\mathrm{T}}\boldsymbol{x}} \right)$$

Optimization

Convex → SGD

Multiclass Classification in Neural Nets



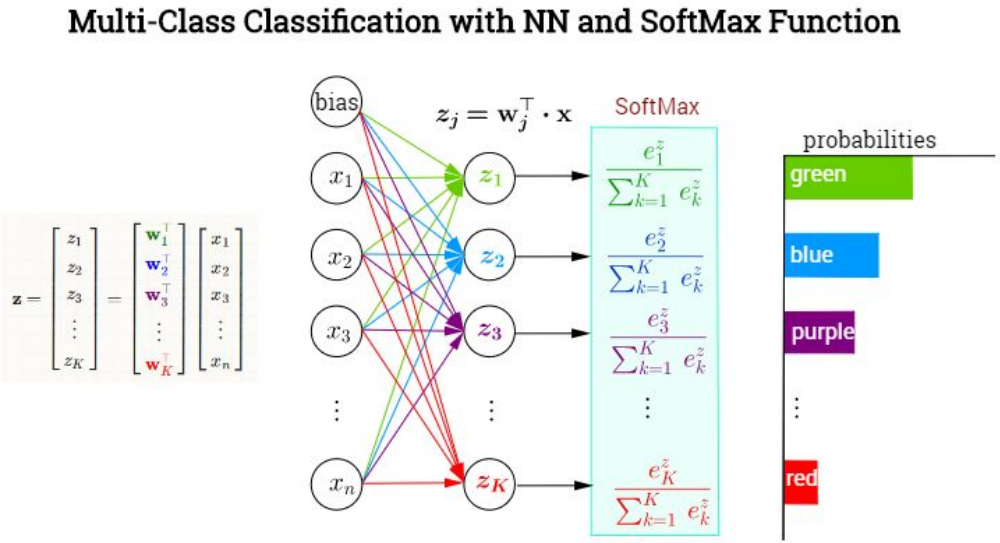


5 separate **binary classifiers**

Key: **sharing the same hidden layers** with **different weights** **at the end**

https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/one-vs-all
http://www.briandolhansky.com/blog/2013/9/23/artificial-neural-nets-linear-multiclass-part-3

Softmax Layer in Neural Nets



$$p(y = j|\mathbf{x}) = \frac{e^{(\mathbf{w}_j^T \mathbf{x} + b_j)}}{\sum_{k \in K} e^{(\mathbf{w}_k^T \mathbf{x} + b_k)}}$$

→ Implementation in PyTorch?

Credit:
https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax

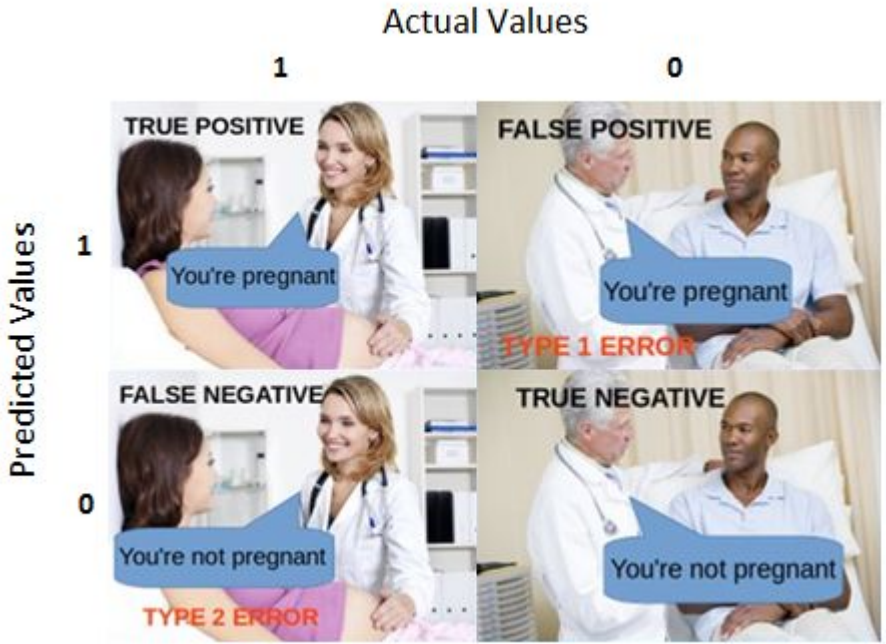
Evaluation: Binary Classifier

- Diagnostic testing table

		True condition			
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$	F_1 score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

Credit: https://en.wikipedia.org/wiki/F-score

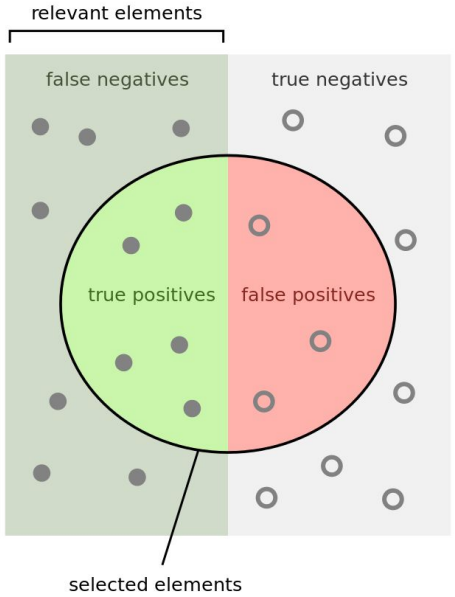
Evaluation: TP, FP, FN, TN & Two Types of Errors

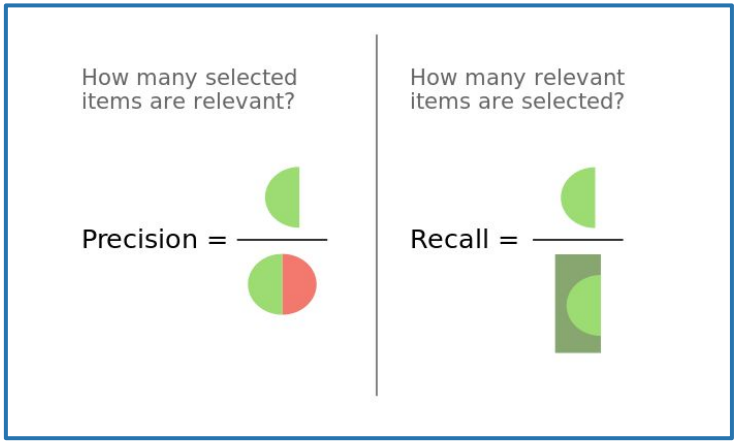


Credit: https://devopedia.org/confusion-matrix

Evaluation: Binary Classifier

- Precision and recall

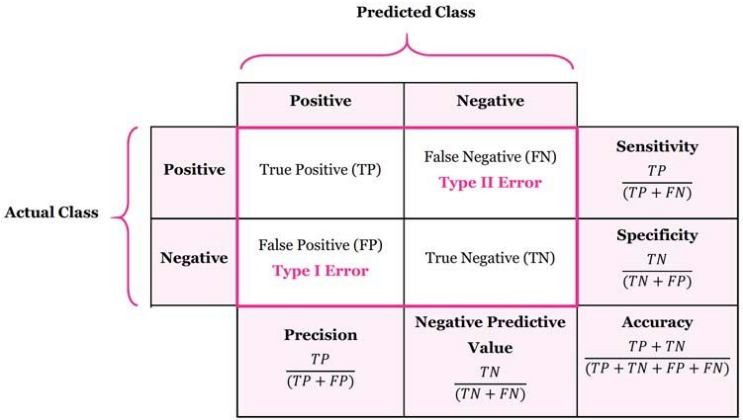




Credit: https://en.wikipedia.org/wiki/F-score

Evaluation: Example

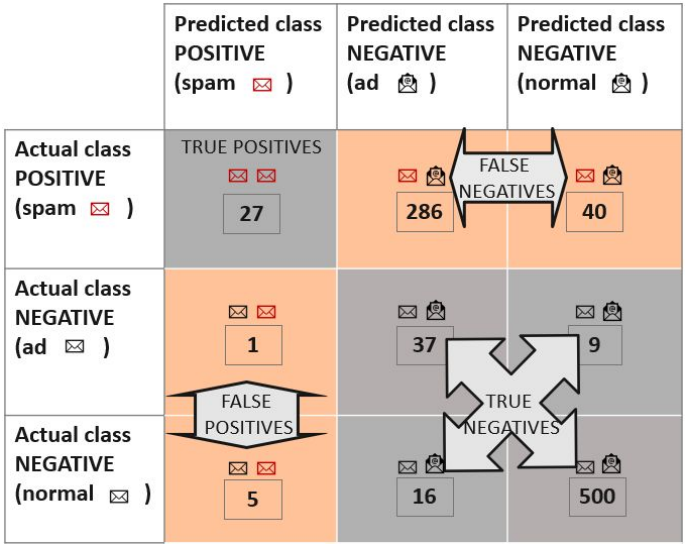
- Calculation from confusion matrix

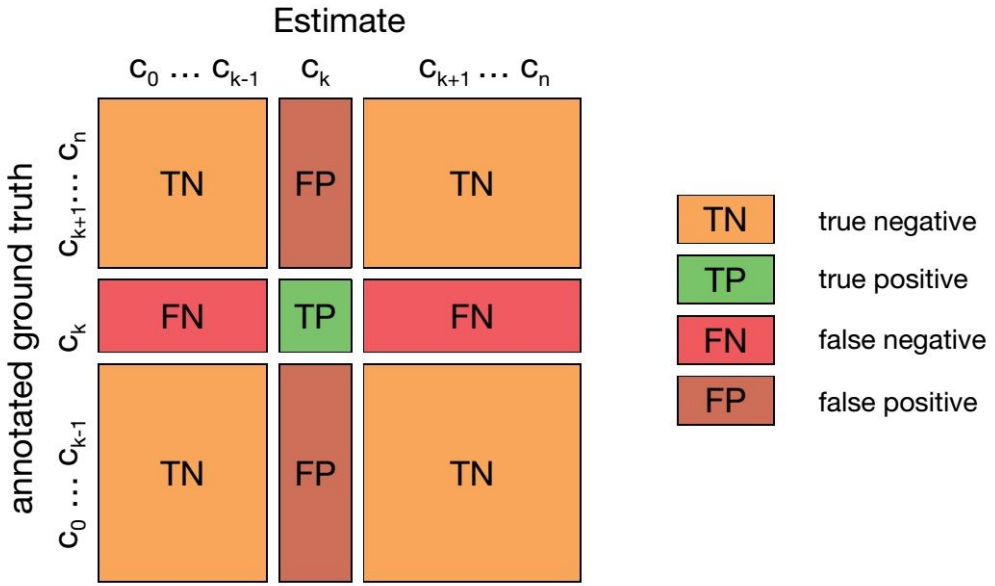


	Predicted class POSITIVE (spam ✉)	Predicted class NEGATIVE (normal 🕵)
Actual class POSITIVE (spam ✉)	TRUE POSITIVE (TP) ✉ ✉ 320	FALSE NEGATIVE (FN) ✉ 🕵 43
Actual class NEGATIVE (normal ✉)	FALSE POSITIVE (FP) ✉ ✉ 20	TRUE NEGATIVE (TN) ✉ 🕵 538

Credit: https://towardsdatascience.com/confusion-matrix-and-class-statistics-68b79f4f510b

Evaluation: Multi-class Confusion Matrix

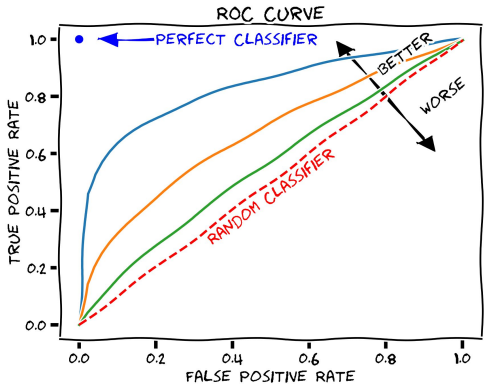




Credit: https://devopedia.org/confusion-matrix

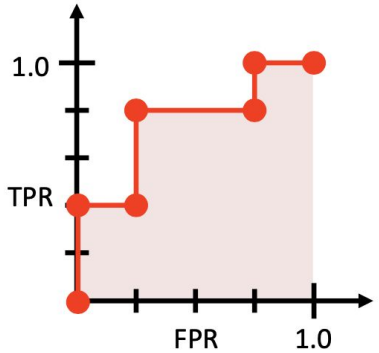
Evaluation: Other Metrics & Curves

- ROC [Wiki]
- Area under the ROC Curve: AUC/AUROC
- A random classifier has AUROC = ½.
- A perfect classifier has AUROC = 1.
- Others: Precision-Recall Curve and AUPR



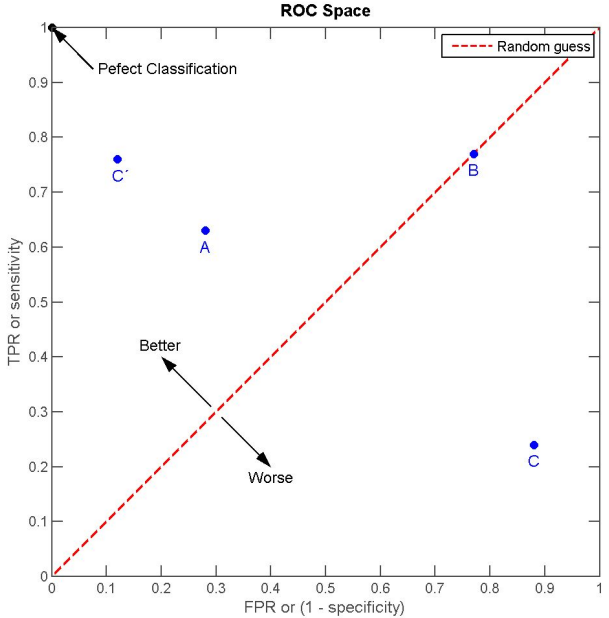
In-class Exercise: Building ROC Curves

Instance	Confidence in positive	Actual class
9	0.99	1
7	0.98	1
1	0.72	0
2	0.70	1
6	0.65	1
10	0.51	0
3	0.39	0
5	0.24	1
4	0.11	0
8	0.01	0

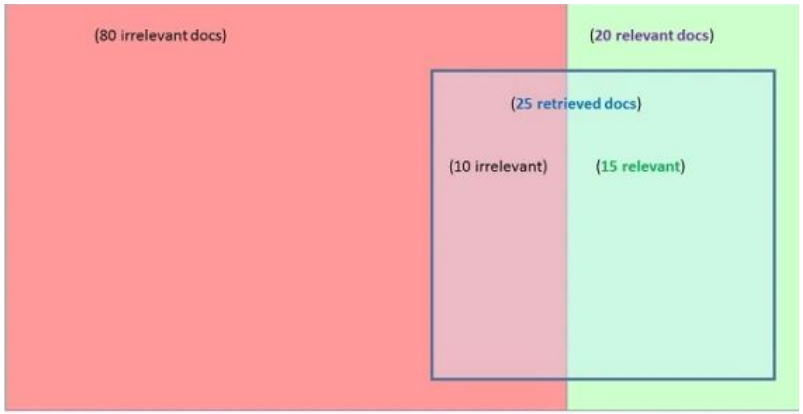


Evaluation: Model Comparison

A			B			C			C′		
TP=63	FP=28	91	TP=77	FP=77	154	TP=24	FP=88	112	TP=76	FP=12	88
FN=37	TN=72	109	FN=23	TN=23	46	FN=76	TN=12	88	FN=24	TN=88	112
100	100	200	100	100	200	100	100	200	100	100	200
TPR = 0.63			TPR = 0.77			TPR = 0.24			TPR = 0.76		
FPR = 0.28			FPR = 0.77			FPR = 0.88			FPR = 0.12		
PPV = 0.69			PPV = 0.50			PPV = 0.21			PPV = 0.86		
F1 = 0.66			F1 = 0.61			F1 = 0.23			F1 = 0.81		
ACC = 0.68			ACC = 0.50			ACC = 0.18			ACC = 0.82		



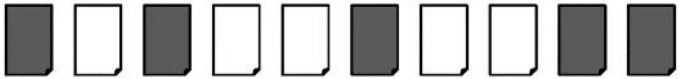
*Precision/Recall in Information Retrieval



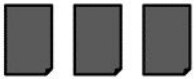
Precision = 15 / 25 = 0.60 Recall = 15 / 20 = 0.75


= relevant documents for query 1

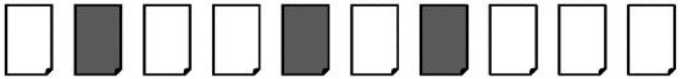
Ranking #1



Recall	0.2	0.2	0.4	0.4	0.4	0.6	0.6	0.6	0.8	1.0
Precision	1.0	0.5	0.67	0.5	0.4	0.5	0.43	0.38	0.44	0.5


= relevant documents for query 2

Ranking #2



Recall	0.0	0.33	0.33	0.33	0.67	0.67	1.0	1.0	1.0	1.0
Precision	0.0	0.5	0.33	0.25	0.4	0.33	0.43	0.38	0.33	0.3

UCLA
Engineer Change.

Evaluation: sklearn Implementation

- Official Doc (stable version): https://scikit-learn.org/stable/modules/model_evaluation.html
- Some helpful demos:
  - Precision-Recall: https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#sphx-glr-auto-examples-model-selection-plot-precision-recall-py
  - Confusion matrix: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html#sklearn.metrics.confusion_matrix
  - Receiver Operating Characteristic (ROC): https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html#sphx-glr-auto-examples-model-selection-plot-roc-py

UCLA
Engineer Change.

Whiteboard

UCLA Samueli Computer Science

Thank you!

Q & A