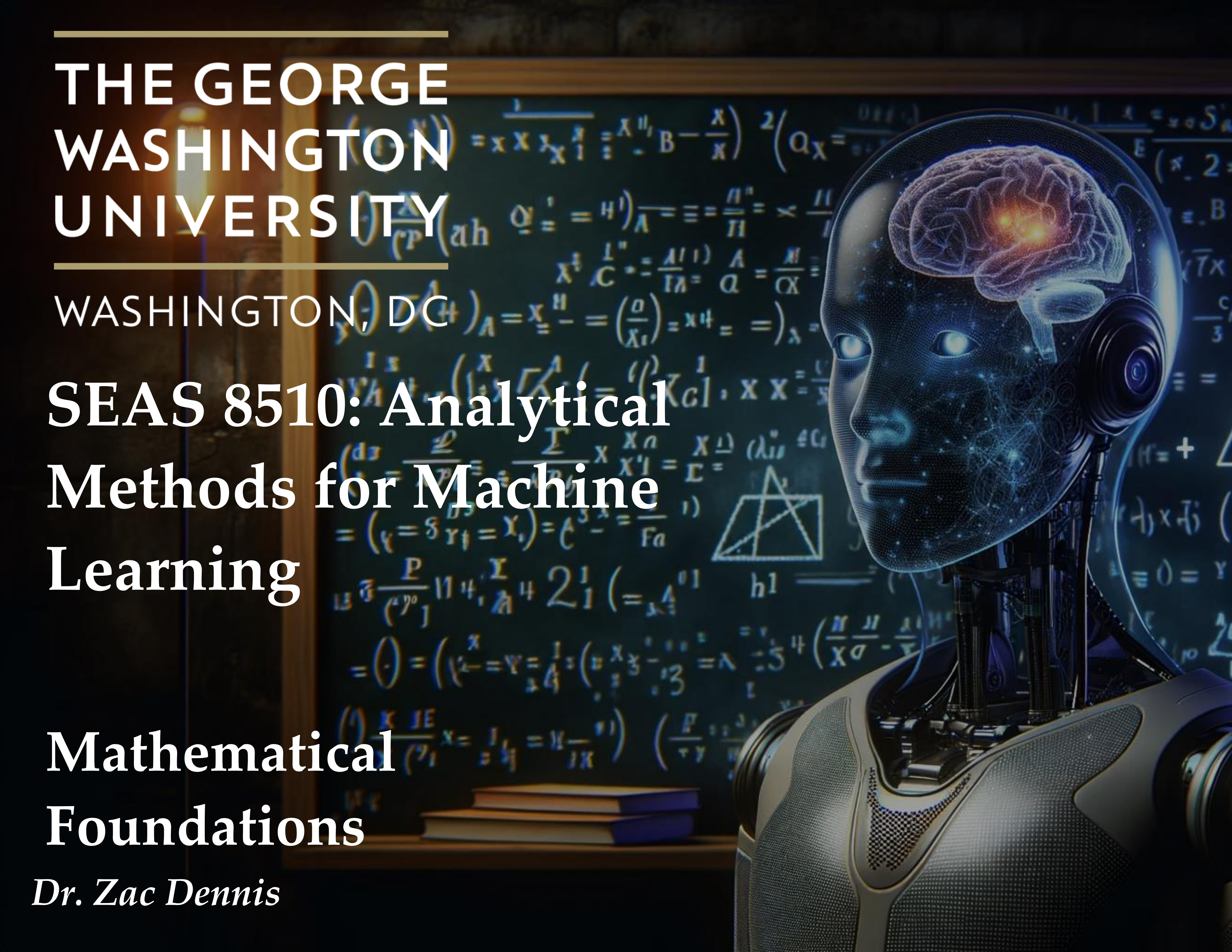
THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

SEAS 8510: Analytical Methods for Machine Learning

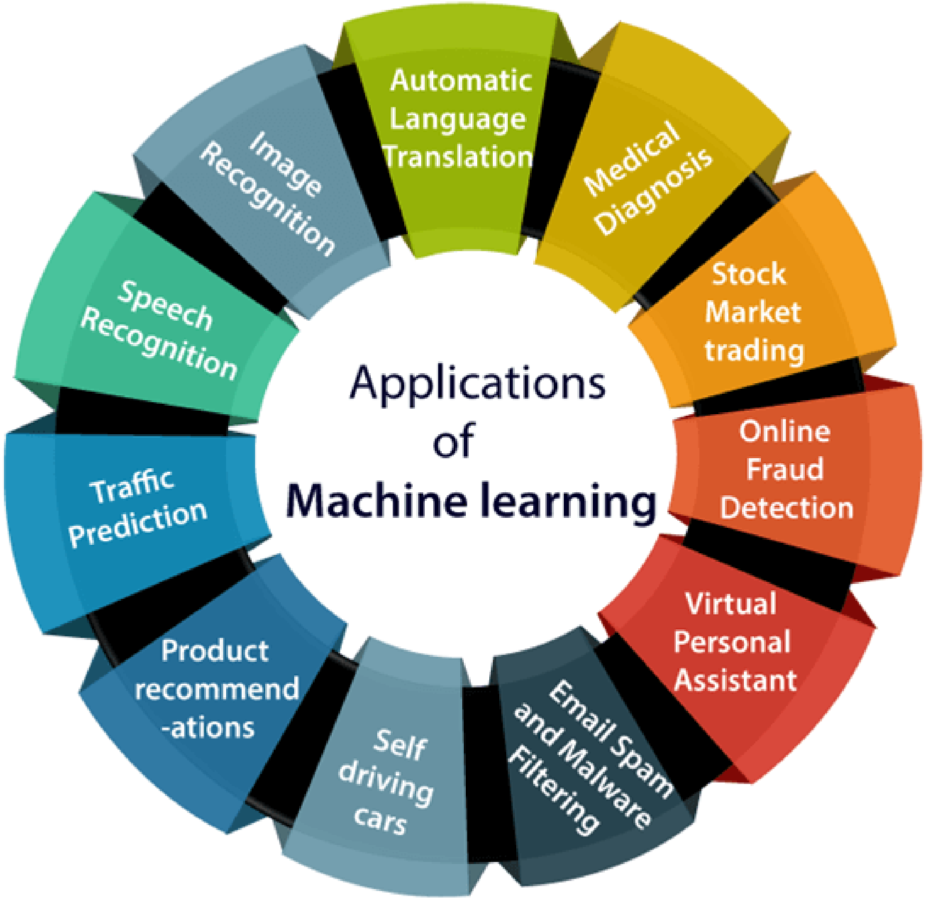
Mathematical Foundations

Dr. Zac Dennis



Machine Learning

- **Definition**: A subset of artificial intelligence (AI) that enables computers to learn from and make decisions based on data.
- **How it Works:**
  - Uses algorithms to find patterns or regularities in data.
  - Adapts automatically through experience.


Applications of Machine learning
Automatic Language Translation
Medical Diagnosis
Stock Market trading
Online Fraud Detection
Virtual Personal Assistant
Email Spam and Malware Filtering
Self driving cars
Product recommend -ations
Traffic Prediction
Speech Recognition
Image Recognition

https://www.javatpoint.com/applications-of-machine-learning

Math and Machine Learning

- The entire process, from data representation to model optimization and validation in machine learning, is governed by mathematical principles and techniques
- **Algorithmic Foundation**: Every machine learning algorithm, from linear regression to deep neural networks, is formulated using mathematical equations and principles.
- **Data Representation**: Data, the core of ML, is represented and manipulated using vectors, matrices, and tensors, concepts from linear algebra.
- **Optimization**: Techniques from calculus are used to adjust and optimize model parameters for better performance, such as in gradient descent.
- **Probabilistic Modeling**: Many ML algorithms, like Naive Bayes and Gaussian Mixture Models, are grounded in probability theory to model uncertainty and make predictions.
- **Statistical Validation**: Evaluating the reliability and significance of ML model results relies on statistical tests and measures.
- **Dimensionality Reduction**: Techniques like Principal Component Analysis (PCA) use eigenvalues and eigenvectors (from linear algebra) to reduce data dimensions while preserving important information.
- **Learning Theory**: Theoretical guarantees about the performance and generalization of algorithms are derived using mathematical proofs and concepts.
- **Regularization**: Mathematical constraints are introduced in models (e.g., L1 and L2 regularization) to prevent overfitting and improve generalization.

Math and ML: Linear Algebra

- **Data Representation**: Vectors represent individual data points, and matrices represent datasets.
- **Transformations**: Eigenvalues and eigenvectors are used in Principal Component Analysis (PCA) for dimensionality reduction.
- **Neural Networks**: Matrix multiplication is employed in the forward and backward propagation processes. Weight matrices and activation vectors define the state and transformation at each layer.
- **Image Processing**: Images are represented as matrices in convolutional neural networks (CNNs), and filters are applied using matrix operations.
- **Recommendation Systems**: Singular Value Decomposition (SVD) factorizes user-item interaction matrices to make recommendations.
- **Support Vector Machines (SVM)**: Dot products between vectors determine the margin between data classes in SVM.
- **Distance Computations**: Euclidean distances between data points, essential for algorithms like k-means clustering and k-nearest neighbors, are calculated using vector operations.
- **Linear Regression**: Model parameters and feature vectors are represented as vectors. The relationship between them is often captured using matrix operations.
- **Natural Language Processing (NLP)**: Word embeddings, like Word2Vec, represent words as vectors, and similarity between words or documents is computed using cosine similarity.

Math and ML: Optimization

- **Gradient Descent:** Used to adjust model parameters to minimize the loss function in algorithms like linear regression, neural networks, and logistic regression.
- **Backpropagation**: Optimization technique in neural networks to adjust weights using the gradient of the error with respect to each weight.
- **Support Vector Machines (SVM):** Quadratic programming is used to find the hyperplane that maximizes the margin between classes.
- **Regularization:** Techniques like L1 (Lasso) and L2 (Ridge) add penalty terms to the loss function, optimizing the trade-off between model complexity and fit to data.
- **Grid Search and Random Search:** Methods to find the best hyperparameters for models by exploring a range of values.
- **Bayesian Optimization:** Probabilistic model-based optimization technique for finding the minimum of functions that are expensive to evaluate.
- **Early Stopping:** A regularization method where training is stopped once model performance stops improving on a held-out validation set.
- **Pruning in Decision Trees:** Reducing the size of decision trees by removing branches that have little power in predicting target values.

Math and ML: Probability

- **Naive Bayes Classifier:** Uses Bayes' theorem to compute the probability of a specific class given the features. Commonly used for text classification.
- **Logistic Regression:** Estimates the probability that a given instance belongs to a particular category using the sigmoid function.
- **Random Forests:** For classification, the probability of each class is estimated based on the proportion of trees voting for that class.
- **Bayesian Neural Networks:** Represents the uncertainty in weights using probability distributions, providing a measure of model uncertainty.
- **Reinforcement Learning:** Uses probabilistic policies to decide actions, and state transitions can be modeled as probabilities.

Math and ML: Statistics

- **Hypothesis Testing:** Used to validate the significance of results, such as determining if differences between algorithms are statistically significant.
- **Linear Regression Analysis:** Uses statistical measures like R^2 and p-values to determine the relationship strength and significance of predictors.
- **Resampling Techniques:** Methods like bootstrapping and cross-validation provide more robust model evaluations and error estimates.
- **Feature Selection:** Techniques like ANOVA and chi-squared tests determine the significance of features for classification tasks.
- **Model Evaluation Metrics:** Metrics such as precision, recall, F1-score, and ROC-AUC are rooted in statistics and are used to evaluate classification model performance.
- **Correlation Analysis:** Measures like Pearson and Spearman correlation coefficients determine the linear relationship between features.
- **Normality Tests:** Tests like the Shapiro-Wilk or Kolmogorov-Smirnov determine if data follows a normal distribution, which influences algorithm choice and data preprocessing.
- **Outlier Detection:** Statistical methods, such as the Z-score or IQR, help identify and handle outliers in datasets.

Advantages of Understanding Math

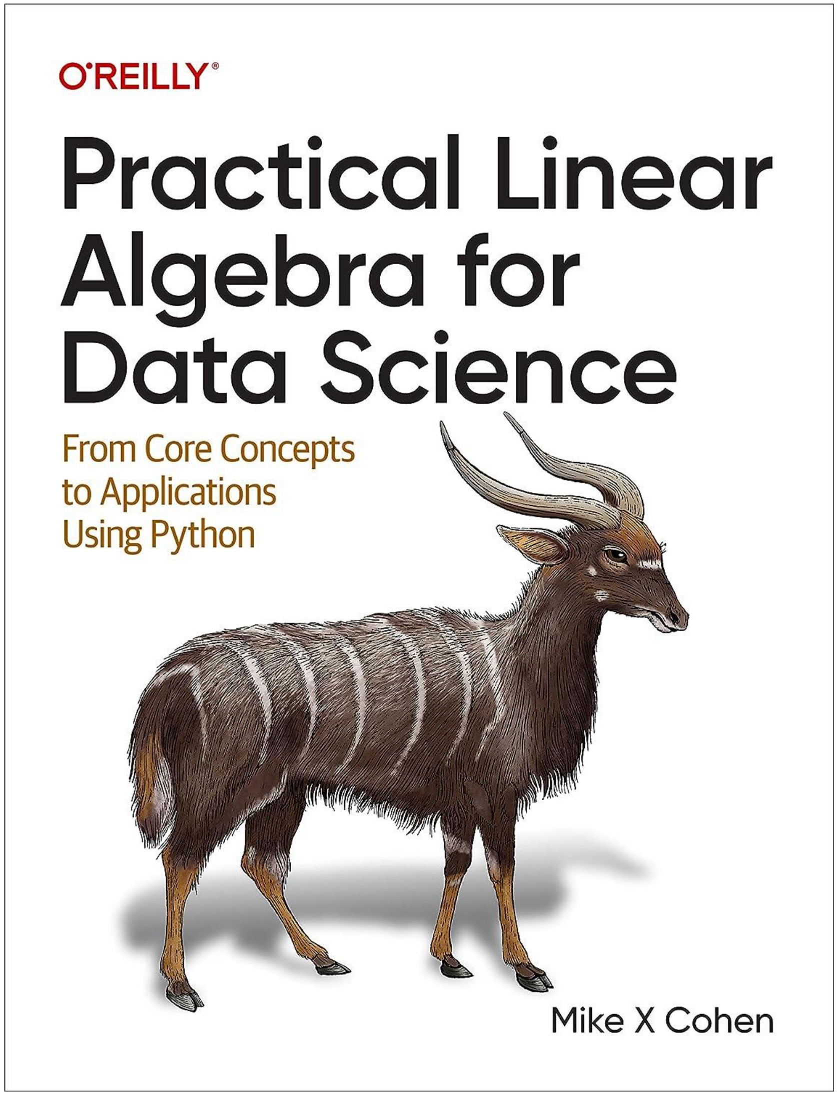
- Deeper understanding of the algorithm's behavior, strengths, and weaknesses. Guide practitioners in choosing the most suitable algorithm for a particular problem
- Selection of appropriate parameter values, leading to better model performance.
- Avoiding overfitting
- Modify or extend algorithms to better suit specific needs
- Efficient debugging
- Interpreting results understanding feature importance
- Keeping up with advances
- More efficient implementations

Linear Algebra

References

- PowerPoint Slides by Dr. Aleksandar (Alex) Vakanski for his course CS 502 Direct Studies: Adversarial Machine Learning https://www.webpages.uidaho.edu/vakanski/CS_502_Fall_2020.html

- Mike X Cohen (2022) Practical Linear Algebra for Data Science. O'Reilly Media, Inc.
- https://wrlc-gwu.primo.exlibrisgroup.com/permalink/01WRLC_GWA/15suu1b/cdi_safari_books_v2_9781098120603


O'REILLY
Practical Linear Algebra for Data Science
From Core Concepts to Applications Using Python
Mike X Cohen

Vectors

- *Vector* definition
  - **Computer science**: *Vector* is a one-dimensional array of ordered real-valued scalars
  - **Mathematics**: *Vector* is a quantity possessing both magnitude and direction, represented by an arrow indicating the direction, and the length of which is proportional to the magnitude
- Vectors are written in column form or in row form
  - Denoted by bold-font lower-case letters

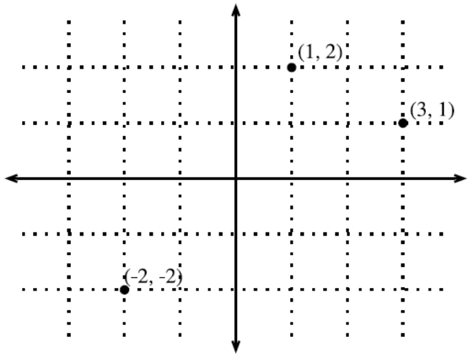
$$\mathbf{x} = \begin{bmatrix} 1 \\ 7 \\ 0 \\ 1 \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} 1 & 7 & 0 & 1 \end{bmatrix}^T$$

- For a general form vector with n elements, the vector lies in the n-dimensional space $\mathbf{x} \in \mathbb{R}^n$

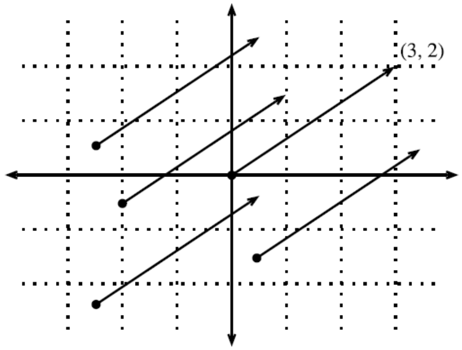
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Geometry of Vectors

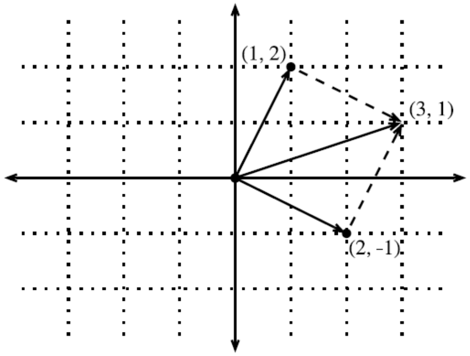
- First interpretation of a vector: point in space
  - E.g., in 2D we can visualize the data points with respect to a coordinate origin



- Second interpretation of a vector: direction in space
  - E.g., the vector $\vec{\mathbf{v}} = [3, 2]^T$ has a direction of 3 steps to the right and 2 steps up
  - The notation $\vec{\mathbf{v}}$ is sometimes used to indicate that the vectors have a direction
  - All vectors in the figure have the same direction



- Vector addition
  - We follow the directions given by the two vectors that are added
- Transpose
  - Converts column vectors into row vectors, and vice versa $\vec{\mathbf{v}}^T_{i,j} = \vec{\mathbf{v}}_{j,i}$



Picture from: http://d2l.ai/chapter_appendix-mathematics-for-deep-learning/geometry-linear-algebraic-ops.html#geometry-of-vectors

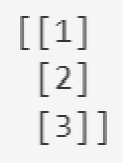
Vectors in Python

```
asList = [1,2,3]
asArray = np.array([1,2,3]) # 1D array
rowVec = np.array([ [1,2,3] ]) # row
colVec = np.array([ [1],[2],[3] ]) # column
```

```
print(f'asList:
{np.shape(asList)}')
print(f'asArray: {asArray.shape}')
print(f'rowVec: {rowVec.shape}')
print(f'colVec: {colVec.shape}')
```

```
asList: (3,)
asArray: (3,)
rowVec: (1, 3)
colVec: (3, 1)
```

```
v=np.array([[1,2,3]])
print(v.T)
```

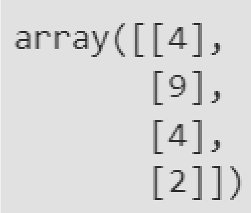


```
[[1]
 [2]
 [3]]
```

Vectors Arithmetic in Python: Broadcasting

```
import numpy as np
Vec1 = np.array([[3],[7],[0],[1] ])
Vec2 = np.array([ [1],[2],[4],[1] ])
Vec1+Vec2
```

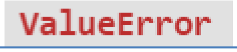
$$\begin{bmatrix}3\\7\\0\\1\end{bmatrix}+\begin{bmatrix}1\\2\\4\\1\end{bmatrix}=\begin{bmatrix}4\\9\\4\\2\end{bmatrix}$$



```
array([[4],
       [9],
       [4],
       [2]])
```

```
import numpy as np
Vec1 = np.array([[3],[7],[0] ])
Vec2 = np.array([ [1],[2],[4],[1] ])
Vec1+Vec2
```

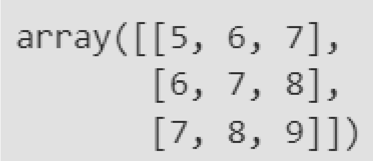
$$\begin{bmatrix}3\\7\\0\end{bmatrix}+\begin{bmatrix}1\\2\\4\\1\end{bmatrix}=?$$



```
ValueError
```

```
import numpy as np
Vec1 = np.array([[1],[2],[3] ])  # column
Vec2 = np.array([ 4,5,6])  # row
Vec1+Vec2
```

$$\begin{bmatrix}1\\2\\3\end{bmatrix}+[4\ 5\ 6]=?$$



```
array([[5, 6, 7],
       [6, 7, 8],
       [7, 8, 9]])
```

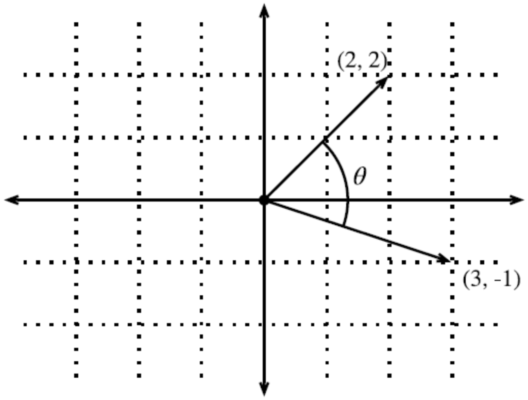
Dot Product and Angles

- *Dot product* of vectors, $\mathbf{u}^T\mathbf{v} = \sum_i u_i \cdot v_i$
  - It is also referred to as inner product, or scalar product of vectors
  - The dot product $\mathbf{u} \cdot \mathbf{v}$ is also often denoted by $\langle \mathbf{u}, \mathbf{v} \rangle$
- The dot product is a symmetric operation, $\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T\mathbf{v} = \mathbf{v}^T\mathbf{u} = \mathbf{v} \cdot \mathbf{u}$

- Geometric interpretation of a dot product: angle between two vectors
  - I.e., dot product $\mathbf{v} \cdot \mathbf{w}$ over the norms of the vectors is $cos(\theta)$

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\|cos(\theta)$$

$$\theta = \arccos\left(\frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}\right)$$



- If two vectors are orthogonal: $\theta = 90°$, i.e., $\cos(\theta) = 0$, then $\mathbf{u} \cdot \mathbf{v} = 0$

Picture from: http://d2l.ai/chapter_appendix-mathematics-for-deep-learning/geometry-linear-algebraic-ops.html#geometry-of-vectors

Norm of a Vector

- A vector ***norm*** is a function that maps a vector to a scalar value
  - The norm is a measure of the size of the vector
- The norm f should satisfy the following properties:
  - Scaling: $f(\alpha \mathbf{x}) = |\alpha| f(\mathbf{x})$
  - Triangle inequality: $f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$
  - Must be non-negative: $f(\mathbf{x}) \geq 0$
- The general ℓ_p norm of a vector $\mathbf{x}$ is obtained as: $\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}$
  - On next page we will review the most common norms, obtained for $p = 1, 2$, and ∞
- Similar concept for matrices is the ***Frobenius norm***
  - It calculates the square-root of the summed squares of the elements of matrix $\mathbf{X}$

$$\|\mathrm{X}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij}^2}$$

Norm of a Vector

- For $p = 2$, we have ℓ_2 norm
  - Also called **Euclidean norm**
  - It is the most often used norm
  - ℓ_2 norm is often denoted just as $\|\mathbf{x}\|$ with the subscript 2 omitted

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{\mathbf{x}^T\mathbf{x}}$$

- For $p = 1$, we have ℓ_1 norm
  - Uses the absolute values of the elements
  - Discriminate between zero and non-zero elements

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$$

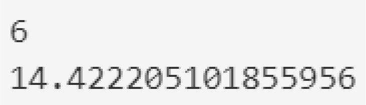
- For $p = \infty$, we have ℓ_∞ norm
  - Known as **infinity norm**, or **max norm**
  - Outputs the absolute value of the largest element

$$\|\mathbf{x}\|_\infty = \max_i |x_i|$$

- ℓ_0 norm outputs the number of non-zero elements
  - It is not a ℓ_p norm, and it is not really a norm function either (it is incorrectly called a norm)

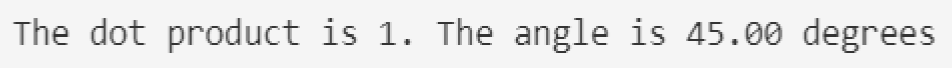
Python: Norm and Dot Product

```
v = np.array([1,2,3,7,8,9])
v_dim = len(v) # math dimensionality
v_mag = np.linalg.norm(v) # math magnitude, length, or norm
print(v_dim,v_mag,sep="\n")
```



```
6
14.422205101855956
```

```
v = np.array([1,0])
w = np.array([1,1])
dot = np.dot(v,w)
angle = np.arccos(dot/(np.linalg.norm(v)*np.linalg.norm(w)))
angle = np.degrees(angle)
print(f"The dot product is {dot}. The angle is {angle:.2f}
degrees",sep="\n")
```



```
The dot product is 1. The angle is 45.00 degrees
```

Correlation

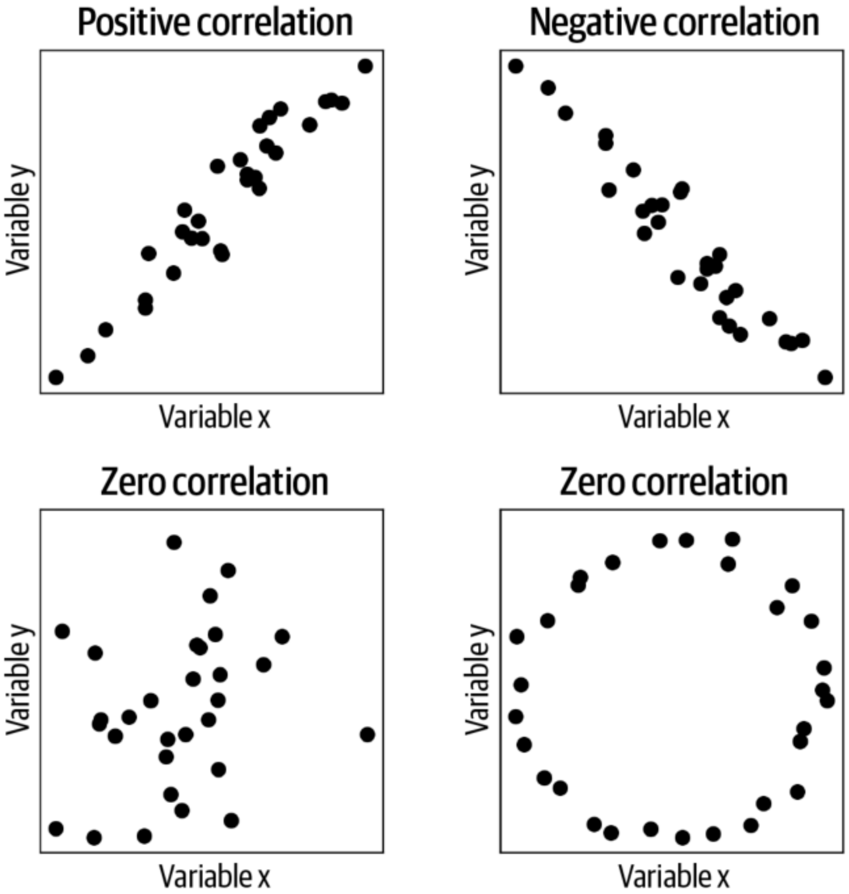
- *Pearson Correlation Coefficient*

$$\rho = \frac{\Sigma_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\Sigma_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\Sigma_{i=1}^{n}(y_i - \bar{y})^2}}$$

- Expressed in linear algebra domain

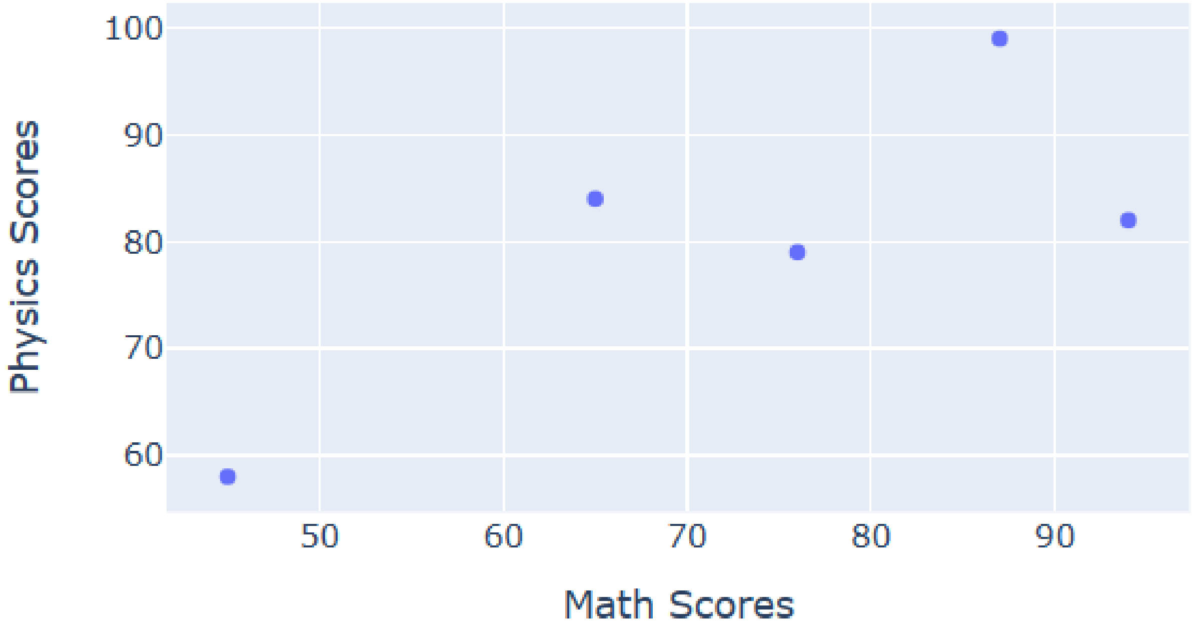
$$\rho = \frac{\widetilde{\mathbf{x}}^{\mathrm{T}}\widetilde{\mathbf{y}}}{\|\widetilde{\mathbf{x}}\| \; \|\widetilde{\mathbf{y}}\|}$$

  - where $\tilde{\mathbf{x}}$ is the mean-centered version of vector $\mathbf{x}$



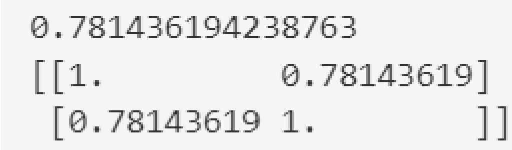
Correlation

```
import plotly.express as px

math = np.array([87, 65, 94, 45, 76])
physics = np.array([99, 84, 82, 58, 79])

# Convert the numpy arrays to lists for Plotly Express compatibility
math_scores = list(math)
physics_scores = list(physics)

# Create a scatter plot
fig = px.scatter(x=math_scores, y=physics_scores, labels={'x': 'Math Scores', 'y': 'Physics    Scores'},
                 title='Scatter Plot of Math vs Physics Scores')

# Show the plot
fig.show()
```



Correlation

```
def pearson_correlation(x, y):
    """
    Compute Pearson correlation coefficient between two vectors.
    """
    # Ensure numpy arrays
    x = np.array(x)
    y = np.array(y)

    # Mean-centering the vectors
    x = x - np.mean(x)
    y = y - np.mean(y)

    # Compute Pearson correlation coefficient
    correlation = np.dot(x, y) / (np.sqrt(np.dot(x, x)) * np.sqrt(np.dot(y, y)))

    return correlation

# Example vectors
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 3, 4, 5, 6])

# Calculate and print the Pearson correlation coefficient
print(pearson_correlation(math, physics))

print(np.corrcoef(math, physics))
```



```
0.781436194238763
[[1.         0.78143619]
 [0.78143619 1.        ]]
```

Correlation

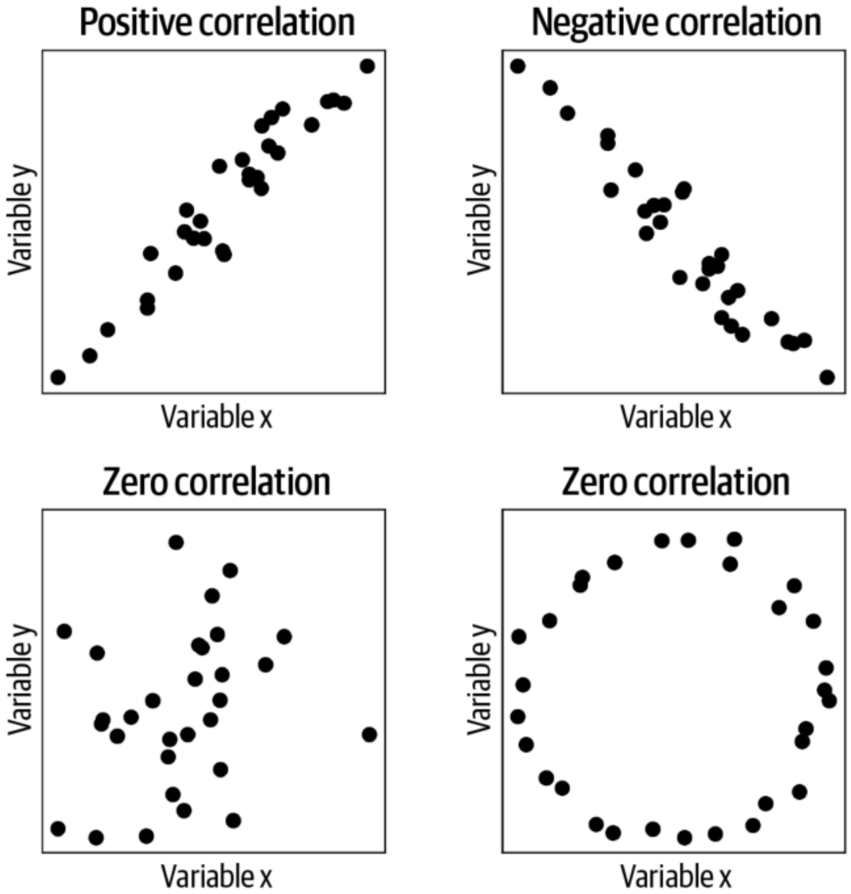
- *Pearson Correlation Coefficient*

$$\rho = \frac{\Sigma_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\Sigma_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\Sigma_{i=1}^{n}(y_i - \bar{y})^2}}$$

- Expressed in linear algebra domain

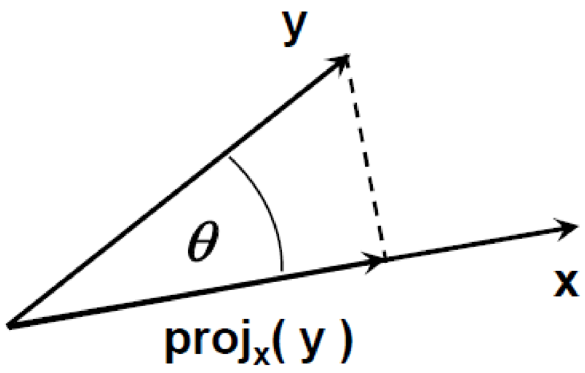
$$\rho = \frac{\tilde{\mathbf{x}}^{\mathrm{T}}\tilde{\mathbf{y}}}{\|\tilde{\mathbf{x}}\| \; \|\tilde{\mathbf{y}}\|}$$

  - where $\tilde{\mathbf{x}}$ is the mean-centered version of vector $\mathbf{x}$



Vector Projection

- *Orthogonal projection* of a vector $\mathbf{y}$ onto vector $\mathbf{x}$
  - The projection can take place in any space of dimensionality ≥ 2
  - The unit vector in the direction of $\mathbf{x}$ is $\frac{\mathbf{x}}{\|\mathbf{x}\|}$
    - A unit vector has norm equal to 1
  - The length of the projection of $\mathbf{y}$ onto $\mathbf{x}$ is $\|\mathbf{y}\| \cdot cos(\theta)$
  - The orthogonal project is the vector $\mathbf{proj_x}(\mathbf{y})$

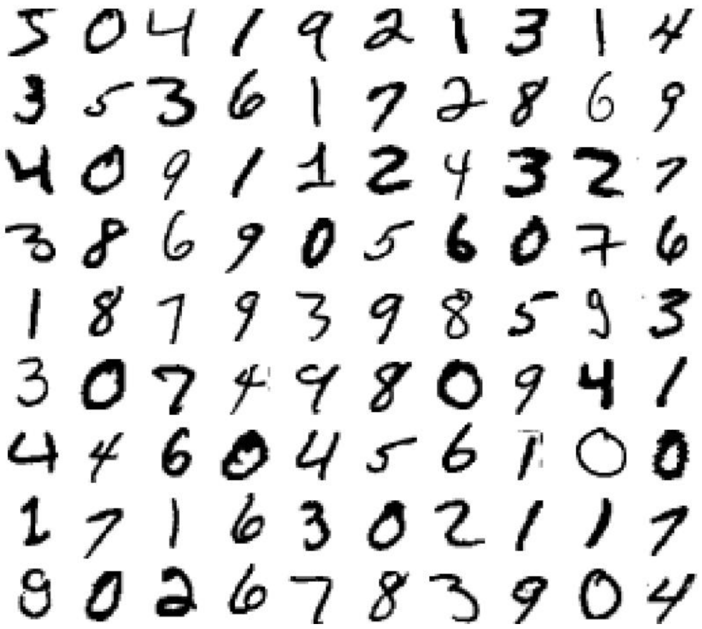


$$\mathbf{proj_x}(\mathbf{y}) = \frac{\mathbf{x}}{\|\mathbf{x}\|} \cdot \|\mathbf{y}\| \cdot cos(\theta)$$

Slide credit: Jeff Howbert — Machine Learning Math Essentials

MNIST

- Set of 70,000 small images of digits handwritten by high school students and employees of the US Census Bureau.
- Each image is 28 × 28 pixels
- Each feature simply represents one pixel’s intensity, from 0 (white) to 255 (black) (i.e. 784 features)
- Each image is labeled with the digit it represents.
- This set has been studied so much that it is often called the “hello world” of Machine Learning: whenever people come up with a new classification algorithm they are curious to see how it will perform on MNIST



Hands-On Machine Learning, Ch3

MNIST – Already Processed

- The MNIST dataset is actually already split into a training set (the first 60,000 images) and a test set (the last 10,000 images):
- The training set is already shuffled for us, which is good because this guarantees that all cross-validation folds will be similar (you don’t want one fold to be missing some digits).

Cautionary Example – Just look, 93% accuracy!

- Above 93% accuracy (ratio of correct predictions) on all cross-validation folds? This looks amazing, doesn’t it? Well, before you get too excited, let’s look at a very dumb classifier that just classifies every single image in the “not-5” class

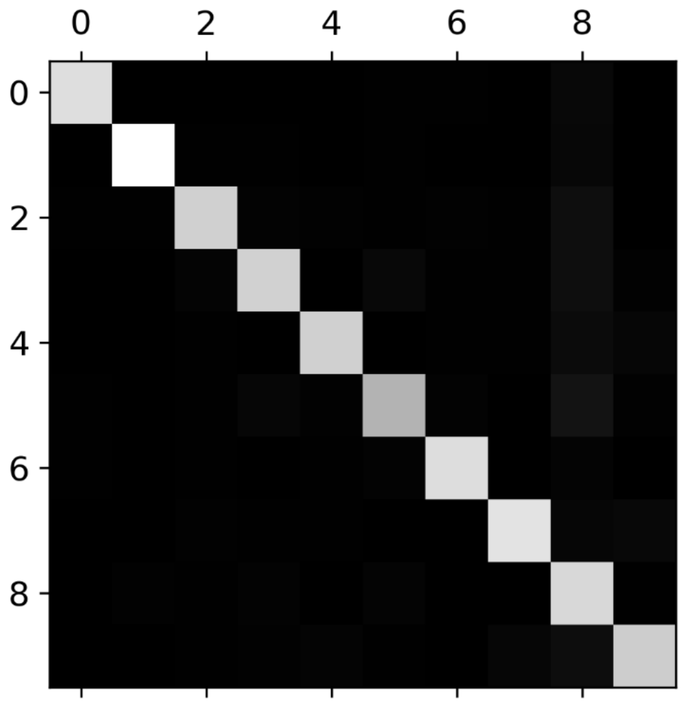
- That’s right, it has over 90% accuracy! This is simply because only about 10% of the images are 5s, so if you always guess that an image is not a 5, you will be right about 90% of the time. Beats Nostradamus.

- This demonstrates why accuracy is generally not the preferred performance measure for classifiers, especially when you are dealing with skewed datasets (i.e., when some classes are much more frequent than others).

Multiclass Classification

- Whereas binary classifiers distinguish between two classes, multiclass classifiers (also called multinomial classifiers) can distinguish between more than two classes.

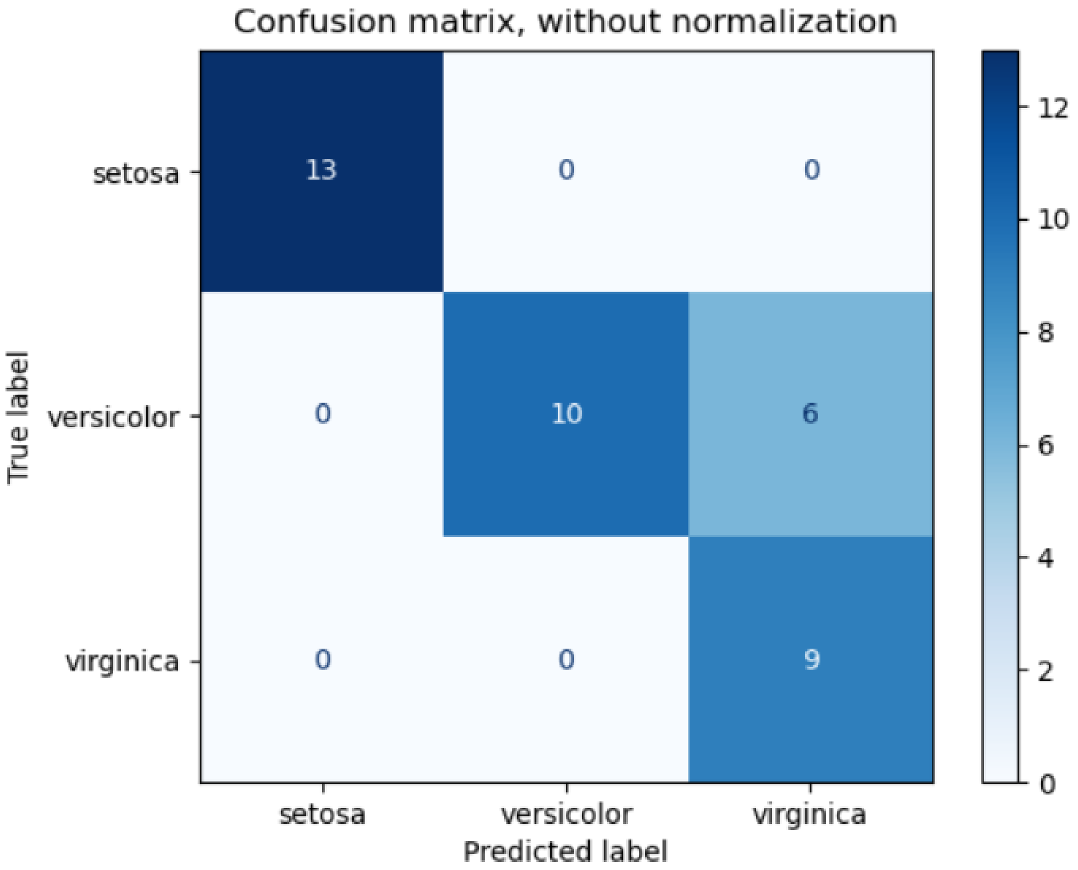
Confusion Matrix



- Better way to evaluate the performance of a classifier
- Count the number of times instances of class A are classified as class B.
  - For example, to know the number of times the classifier confused images of 5s with 3s, you would look in the fifth row and third column of the confusion matrix.
- Each row in a confusion matrix represents an actual class, while each column represents a predicted class.
- A perfect classifier would have only true positives and true negatives, so its confusion matrix would have nonzero values only on its main diagonal (top left to bottom right)

Confusion Matrix and Common Outputs

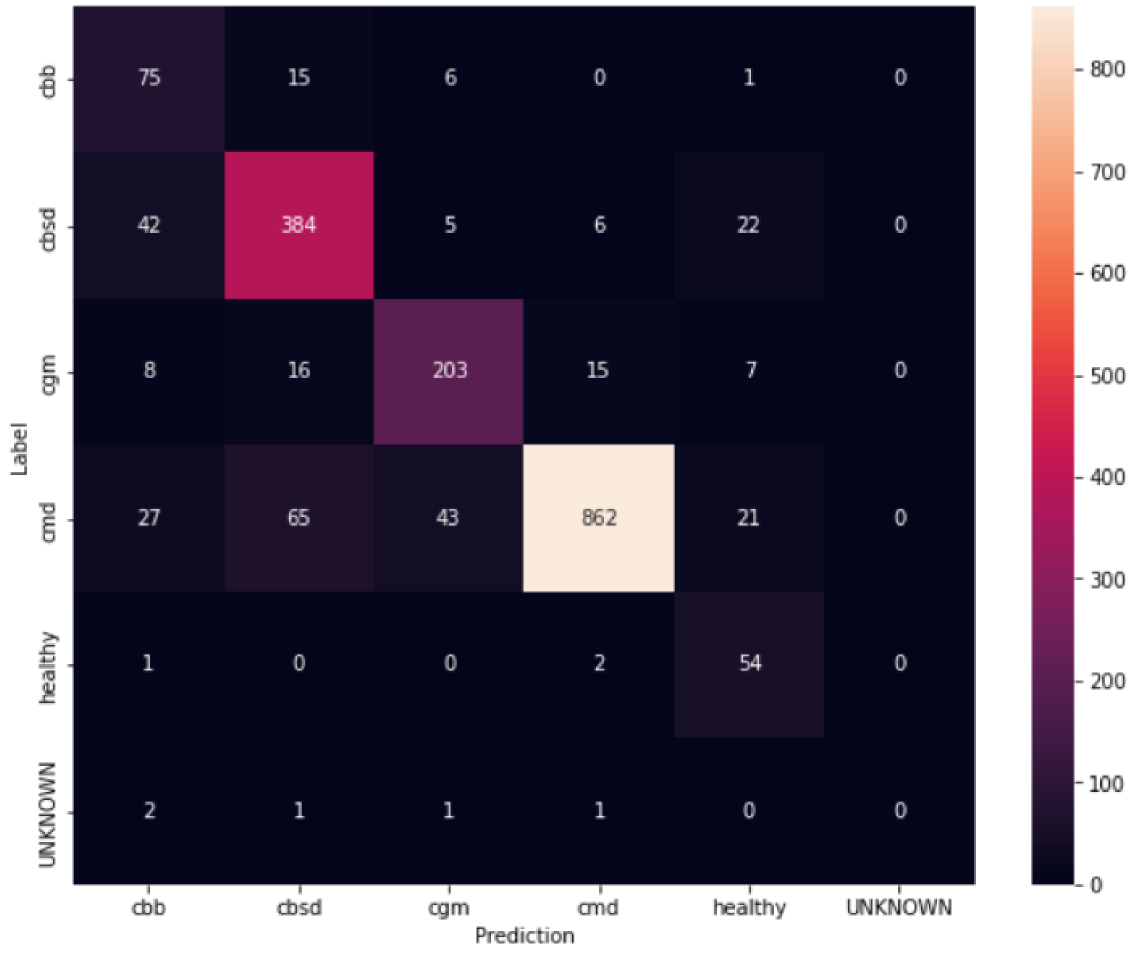
- Confusion matrix used to evaluate a classifier
- Diagonals are when the predicted label is equal to the true label
- Off-diagonal elements are those that are mislabeled by the classifier



https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py

TF Tutorial: Fine Tuning Models for Plant Disease Detection

- Enriches the data with unknown (negative) examples to get a more robust model
- Applies image augmentation
- Uses a confusion matrix for classifier output

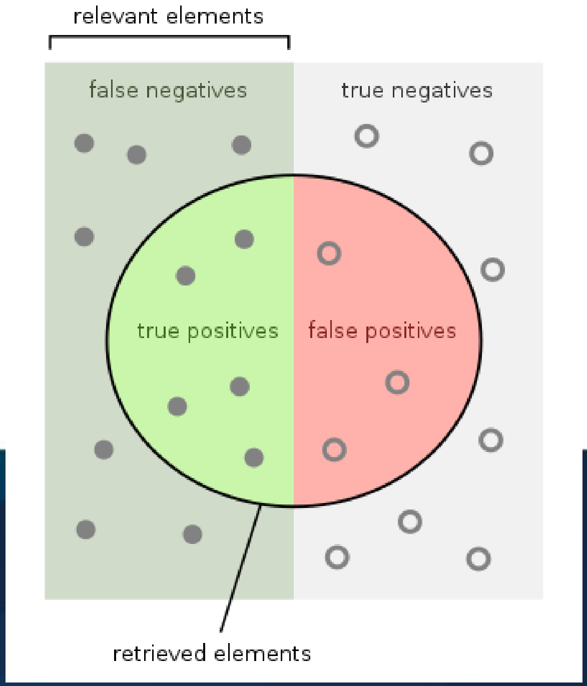


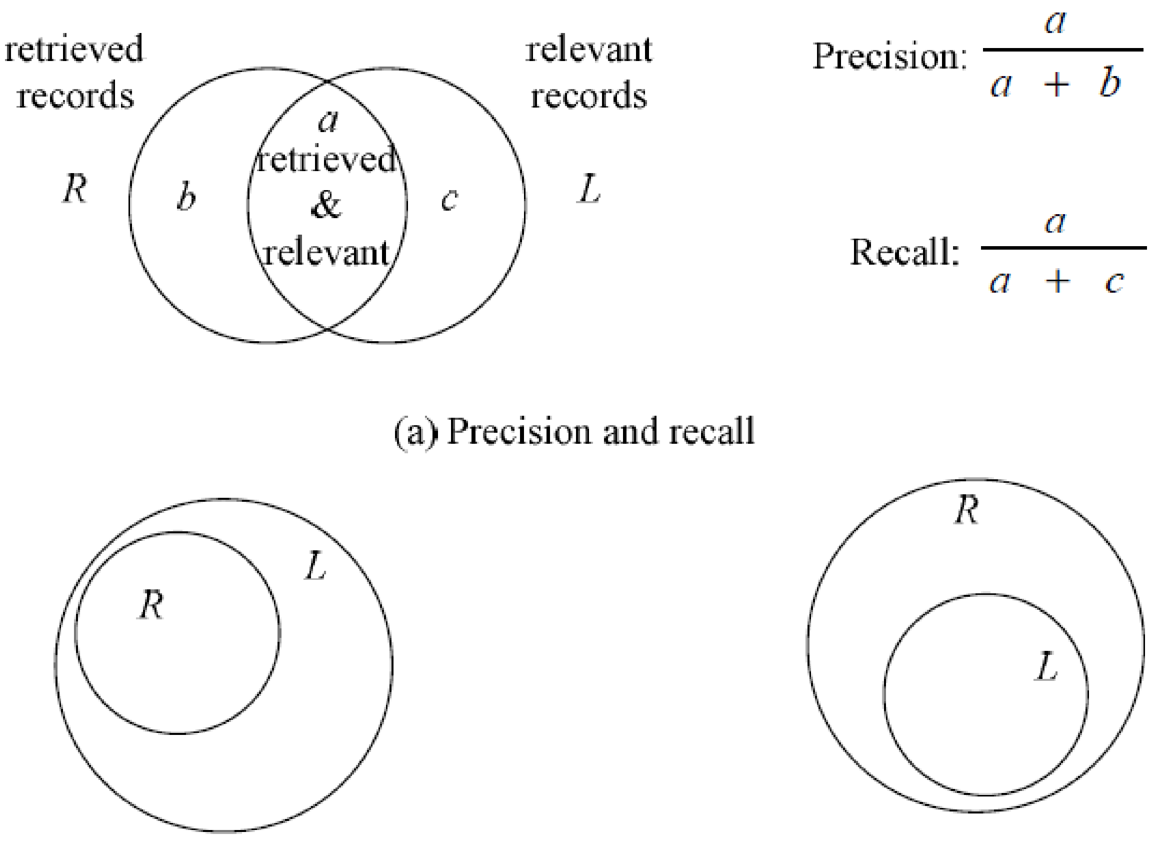
Label \ Prediction	cbb	cbsd	cgm	cmd	healthy	UNKNOWN
cbb	75	15	6	0	1	0
cbsd	42	384	5	6	22	0
cgm	8	16	203	15	7	0
cmd	27	65	43	862	21	0
healthy	1	0	0	2	54	0
UNKNOWN	2	1	1	1	0	0

https://www.tensorflow.org/hub/tutorials/cropnet_on_device

Precision and Recall

- **Precision**: ratio of true positives divided by the total number of positive predictions
- **Recall**: ratio of true positives divided by the total number of positive examples





Precision: $\frac{a}{a+b}$

Recall: $\frac{a}{a+c}$

(a) Precision and recall

(b) Precision = 1

(c) Recall = 1

Precision

$$\text{precision} = \frac{TP}{TP + FP}$$

- Accuracy of the positive predictions; this is called the precision of the classifier
- TP is the number of true positives, and FP is the number of false positives

Recall

$$\text{recall} = \frac{TP}{TP + FN}$$

- Precision is typically used along with another metric named recall, also called sensitivity or the true positive rate (TPR): this is the ratio of positive instances that are correctly detected by the classifier
- FN is the number of false negatives

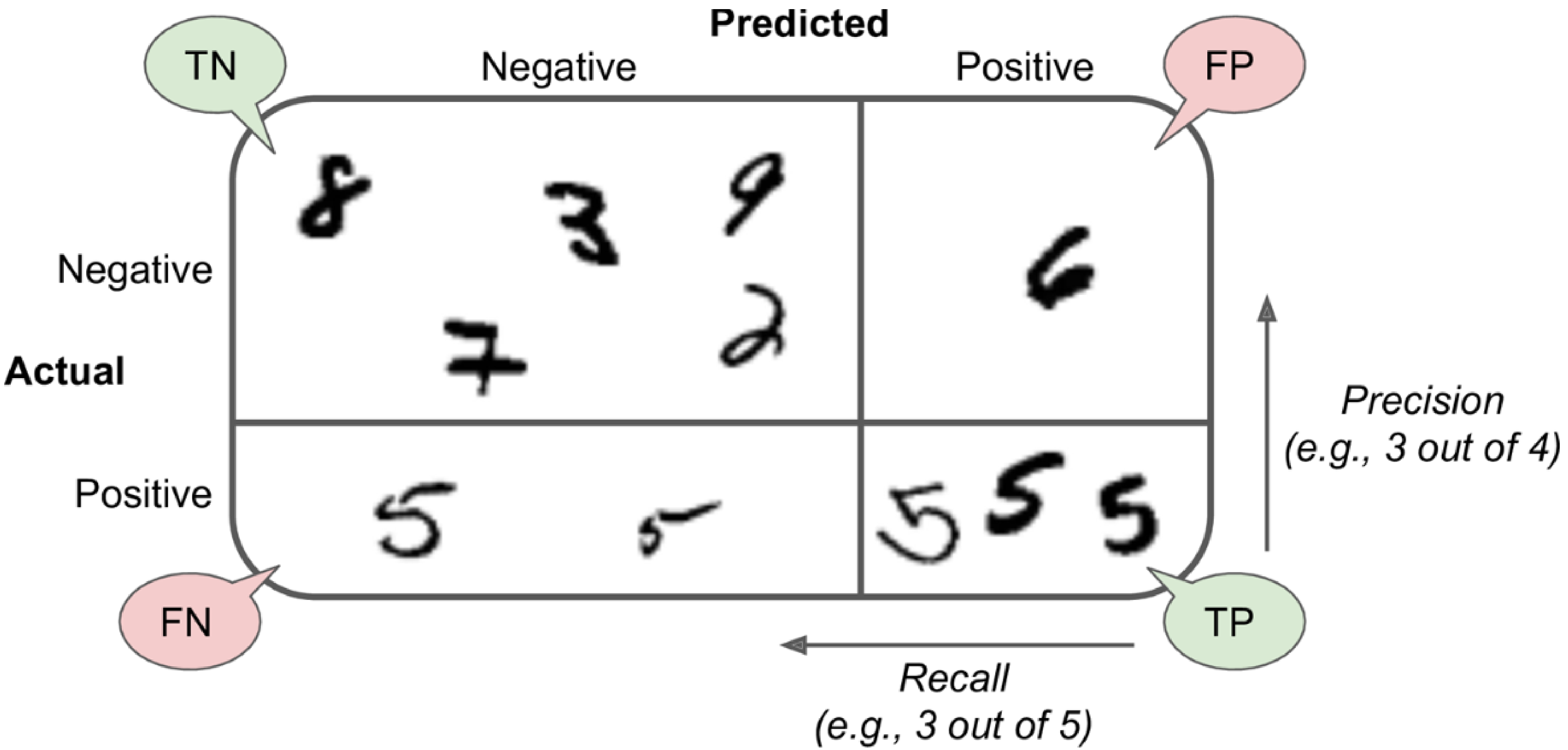


Figure 3-2. An illustrated confusion matrix shows examples of true negatives (top left), false positives (top right), false negatives (lower left), and true positives (lower right)

Precision/Recall Trade-off

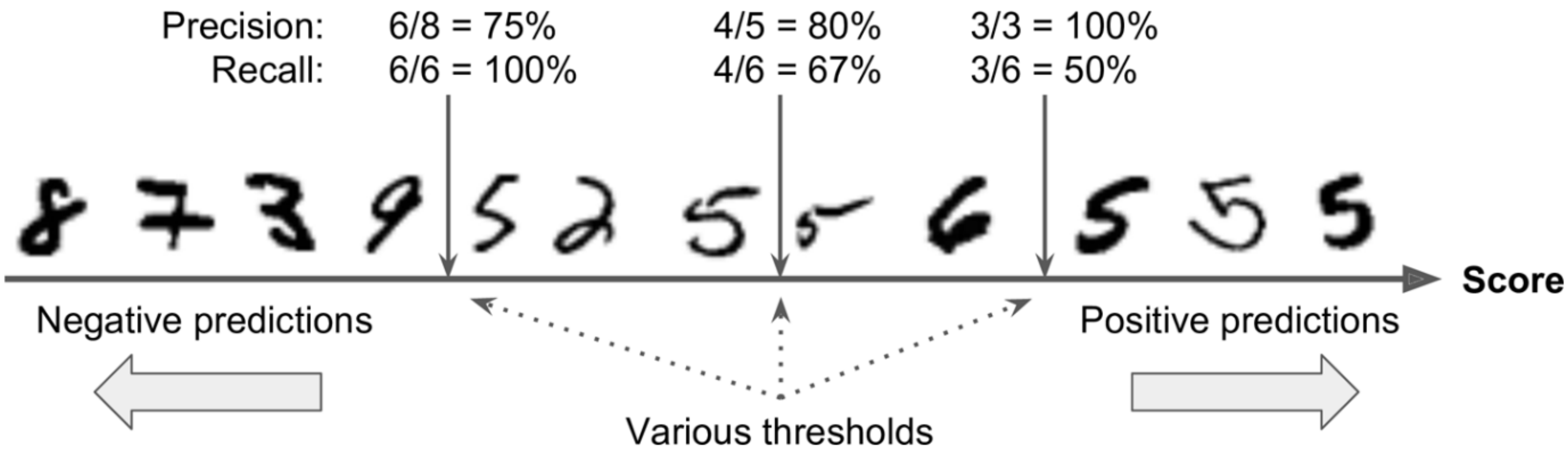


Figure 3-3. In this precision/recall trade-off, images are ranked by their classifier score, and those above the chosen decision threshold are considered positive; the higher the threshold, the lower the recall, but (in general) the higher the precision

- Increasing precision reduces recall, and vice versa.
- Raising the threshold decreases recall.

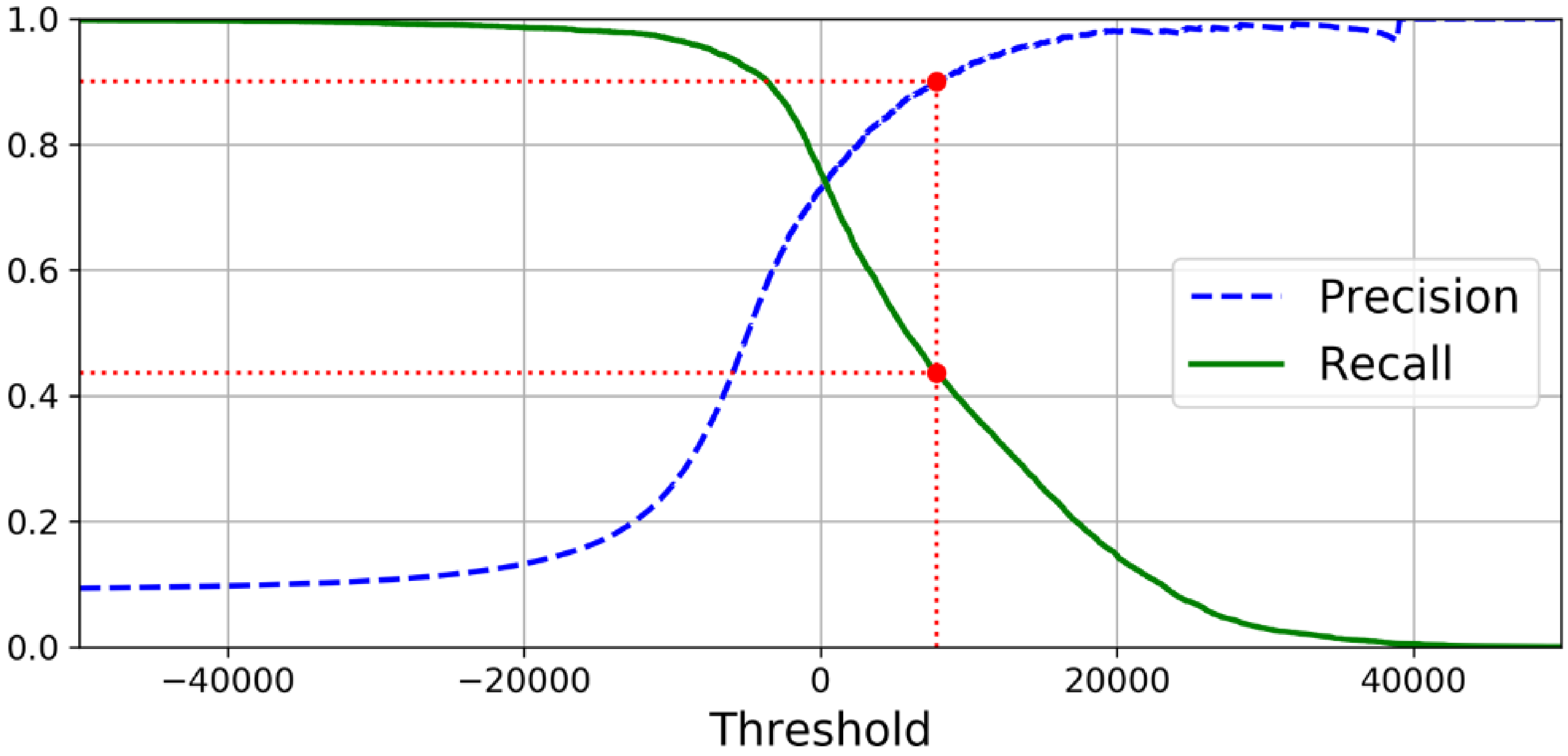
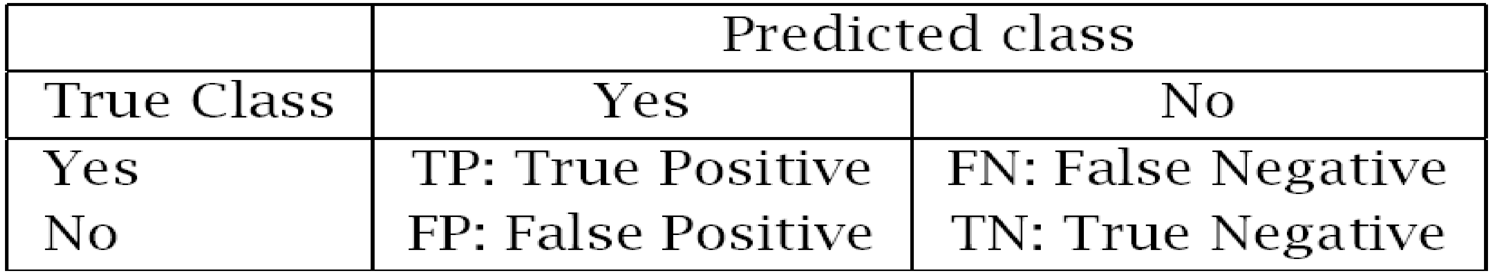


Figure 3-4. Precision and recall versus the decision threshold

Performance Measures



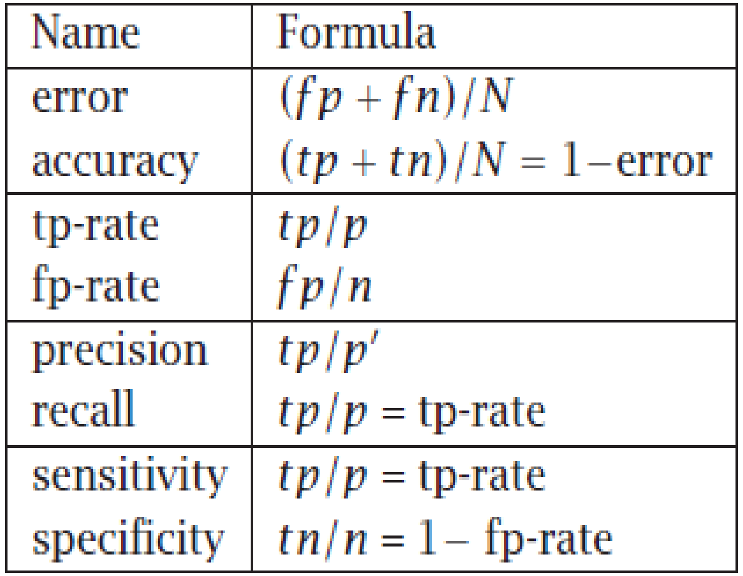
	Predicted class	
True Class	Yes	No
Yes	TP: True Positive	FN: False Negative
No	FP: False Positive	TN: True Negative

- Error rate = # of errors / # of instances = (FN+FP) / N
- Recall = # of found positives / # of positives
- = TP / (TP+FN) = sensitivity = hit rate
- Precision = # of found positives / # of found
- = TP / (TP+FP)
- Specificity = TN / (TN+FP)
- False alarm rate = FP / (FP+TN) = 1 - Specificity

$$\text{accuracy} \overset{\text{def}}{=} \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Accuracy, Sensitivity, Specificity

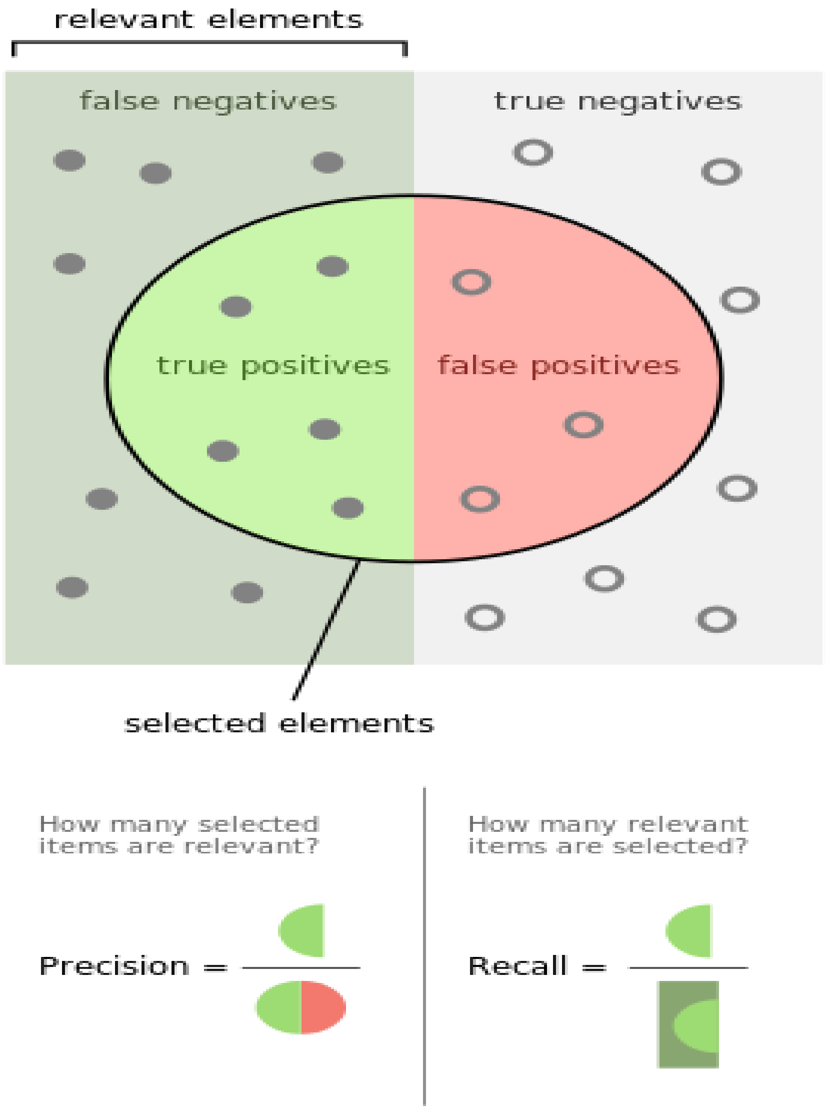
Accuracy: total number of correctly classified examples divided by the total number of classified examples



Name	Formula
error	$(fp+fn)/N$
accuracy	$(tp+tn)/N = 1-\text{error}$
tp-rate	tp/p
fp-rate	fp/n
precision	tp/p'
recall	$tp/p = \text{tp-rate}$
sensitivity	$tp/p = \text{tp-rate}$
specificity	$tn/n = 1-\text{fp-rate}$

$$\text{TPR} \overset{\text{def}}{=} \frac{\text{TP}}{\text{TP}+\text{FN}} \quad \text{and} \quad \text{FPR} \overset{\text{def}}{=} \frac{\text{FP}}{\text{FP}+\text{TN}}$$

MLE, Chapter 5



By Walber - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=36926283

F-Measure

- Percentage of retrieved documents that are relevant: **precision**=TP/(TP+FP)

- Percentage of relevant documents that are returned:
  recall =TP/(TP+FN)

- F-measure=(2 × recall × precision)/(recall+precision)

- Sometimes sensitivity × specificity is used as a measure:

- Sensitivity x Specificity = TP-rate x (1 – FP-rate) = (TP / (TP + FN)) × (TN / (FP + TN))

F1 Score

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN+FP}{2}}$$

- Simple way to compare two classifiers
- Combines precision and recall into a single metric
- The classifier will only get a high F1 score if both recall and precision are high
- The F1 score favors classifiers that have similar precision and recall

The ROC Curve

- The receiver operating characteristic (ROC) curve is another common tool used with binary classifiers.
- Very similar to the precision/recall curve however the ROC curve plots the true positive rate (another name for recall) against the false positive rate (FPR).
- The FPR is the ratio of negative instances that are incorrectly classified as positive.
- The TNR is also called specificity. Hence, the ROC curve plots sensitivity (recall) versus 1 – specificity.
- Can compare classifiers by measuring the area under the curve (AUC).
  - A perfect classifier will have a ROC AUC equal to 1,
  - A purely random classifier will have a ROC AUC equal to 0.5.

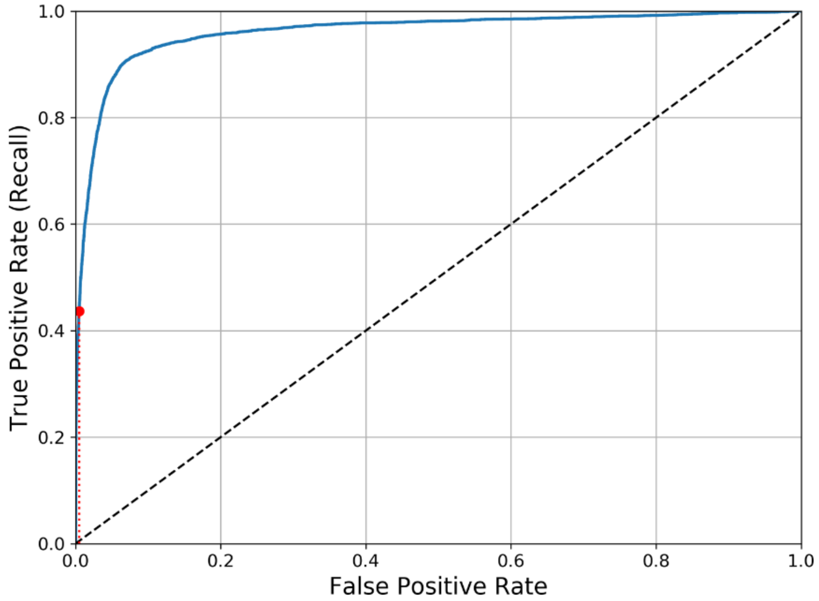


Figure 3-6. This ROC curve plots the false positive rate against the true positive rate for all possible thresholds; the red circle highlights the chosen ratio (at 43.68% recall)

ROC Curve

- Area under the ROC curve (AUC):
Is the probability that randomly chosen positive instance is ranked above randomly chosen negative one

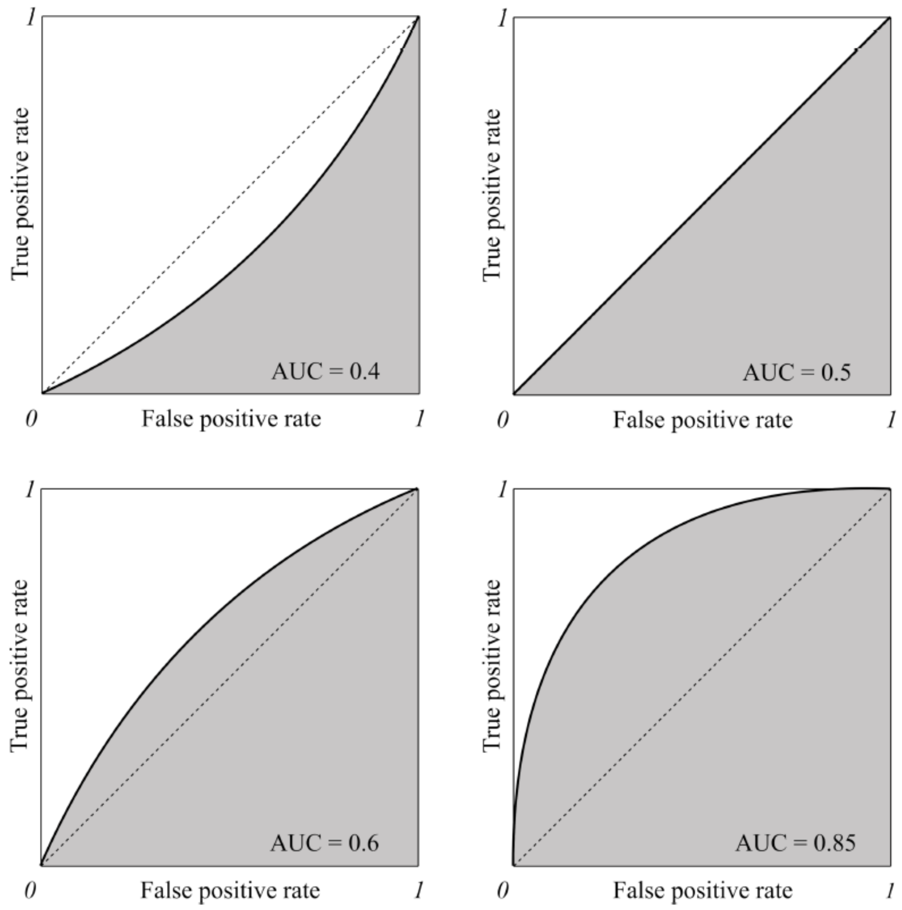
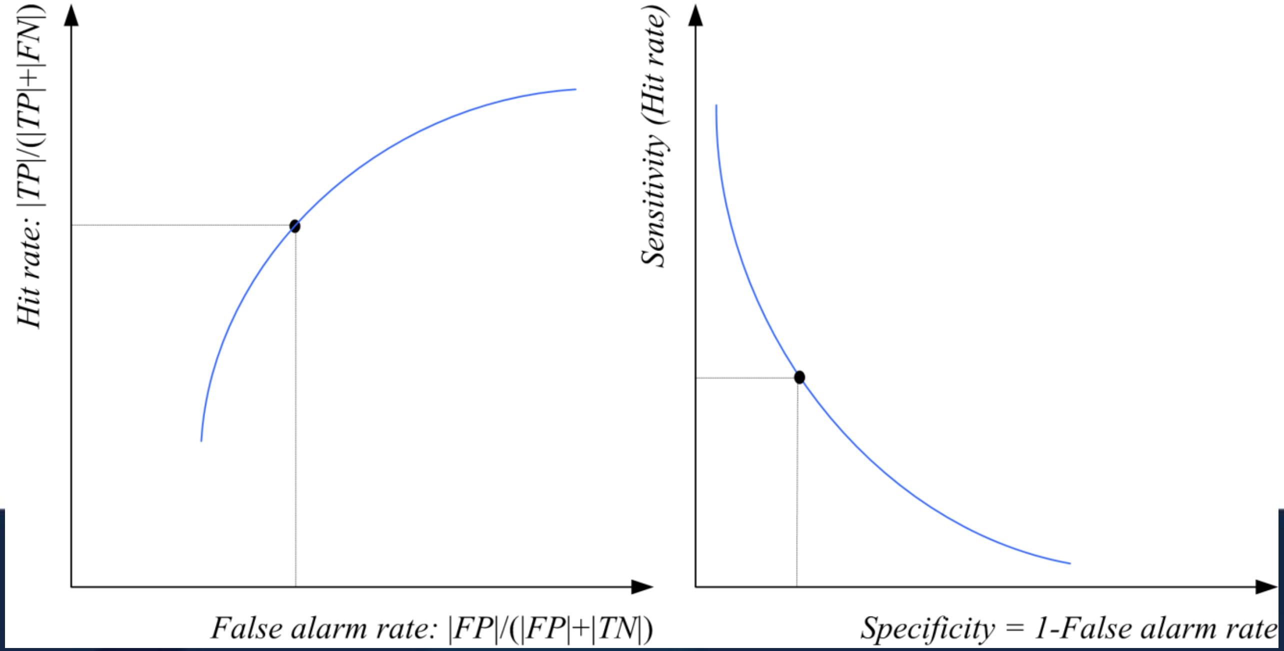
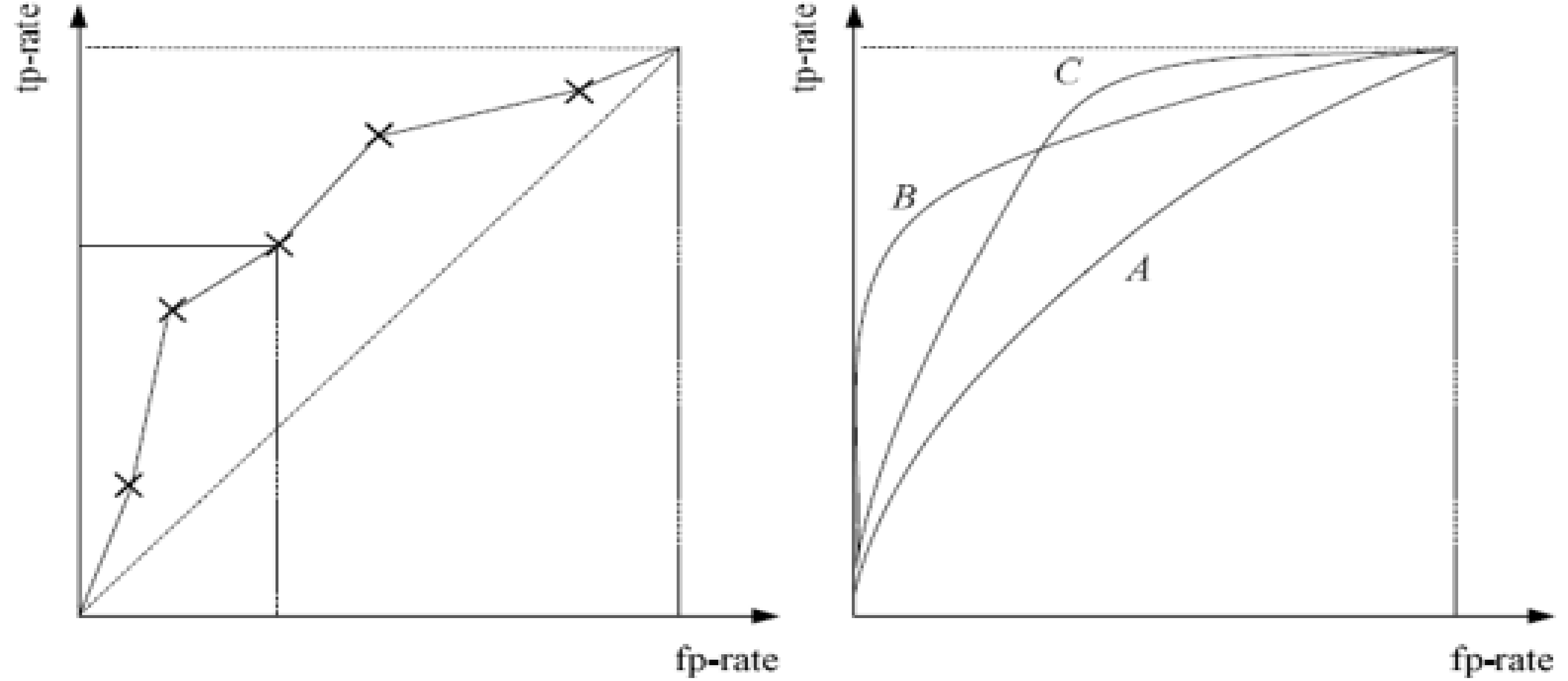


Figure 5: The area under the ROC curve (shown in grey).

MLE, p145



Example ROC Curves

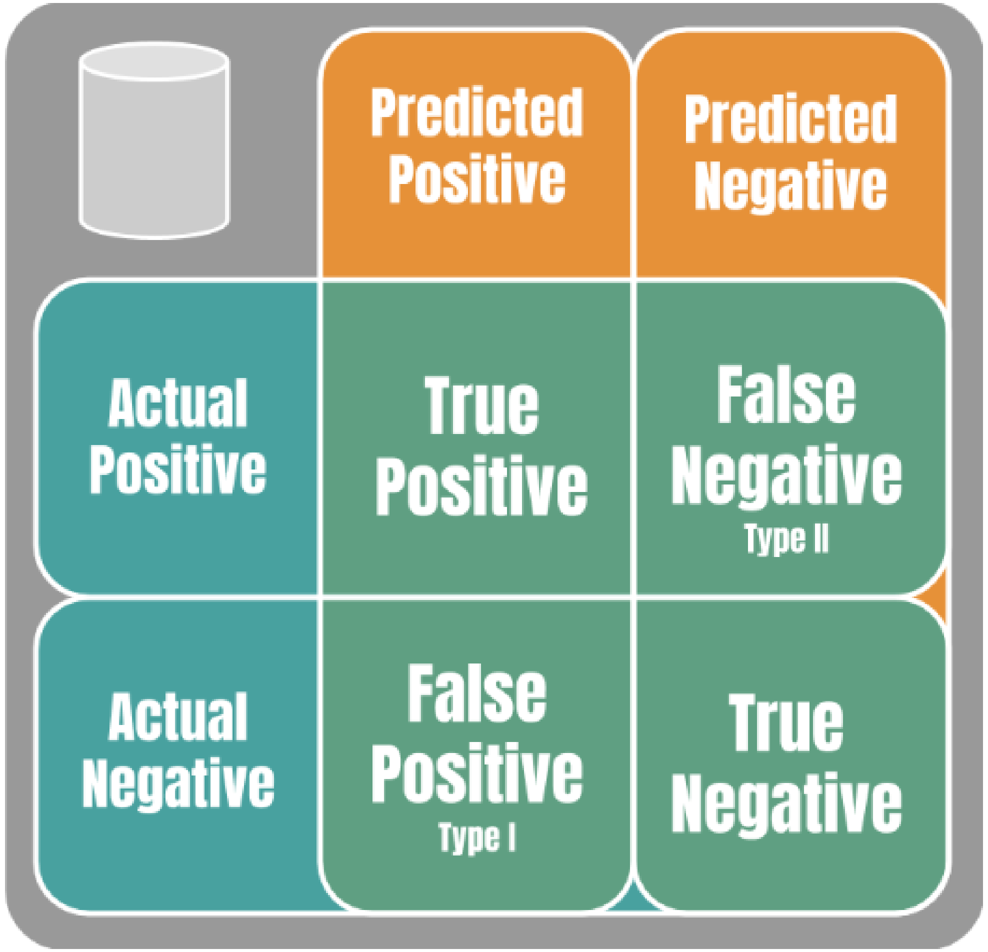


(a) Typical ROC curve. Each classifier has a threshold that allows us to move over this curve, and we decide on a point, based on the relative importance of hits versus false alarms, namely, true positives and false positives. The area below the ROC curve is called AUC.

(b) A classifier is preferred if its ROC curve is closer to the upper-left corner (larger AUC). B and C are preferred over A; B and C are preferred under different loss matrices.

Common Useage of Type I, Type II Errors in Machine Learning

- Can use these within risk management
- Help support the need for additional data collection, augmentation, etc.
- Class construction to align with task and business needs (e.g. modify specificity such that errors are reduced)
- Consequences of Type I and II errors can vary widely across industries
    - Reduced sales for advertising
    - Denying qualified candidates for bank loans
    - Self-driving cars and collisions



	Predicted Positive	Predicted Negative
Actual Positive	True Positive	False Negative Type II
Actual Negative	False Positive Type I	True Negative

https://towardsdatascience.com/machine-learning-an-error-by-any-other-name-a7760a702c4d