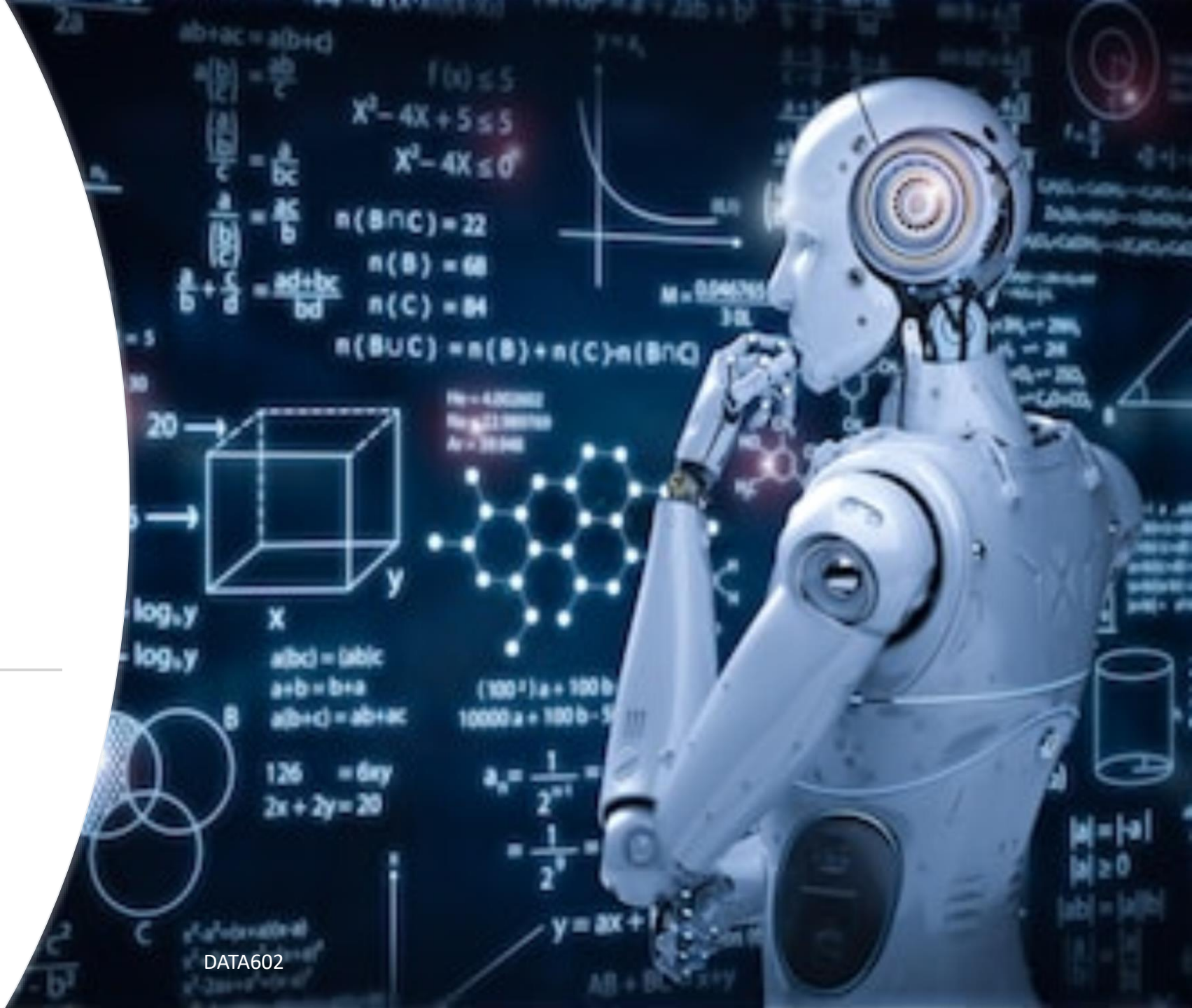
UMBC

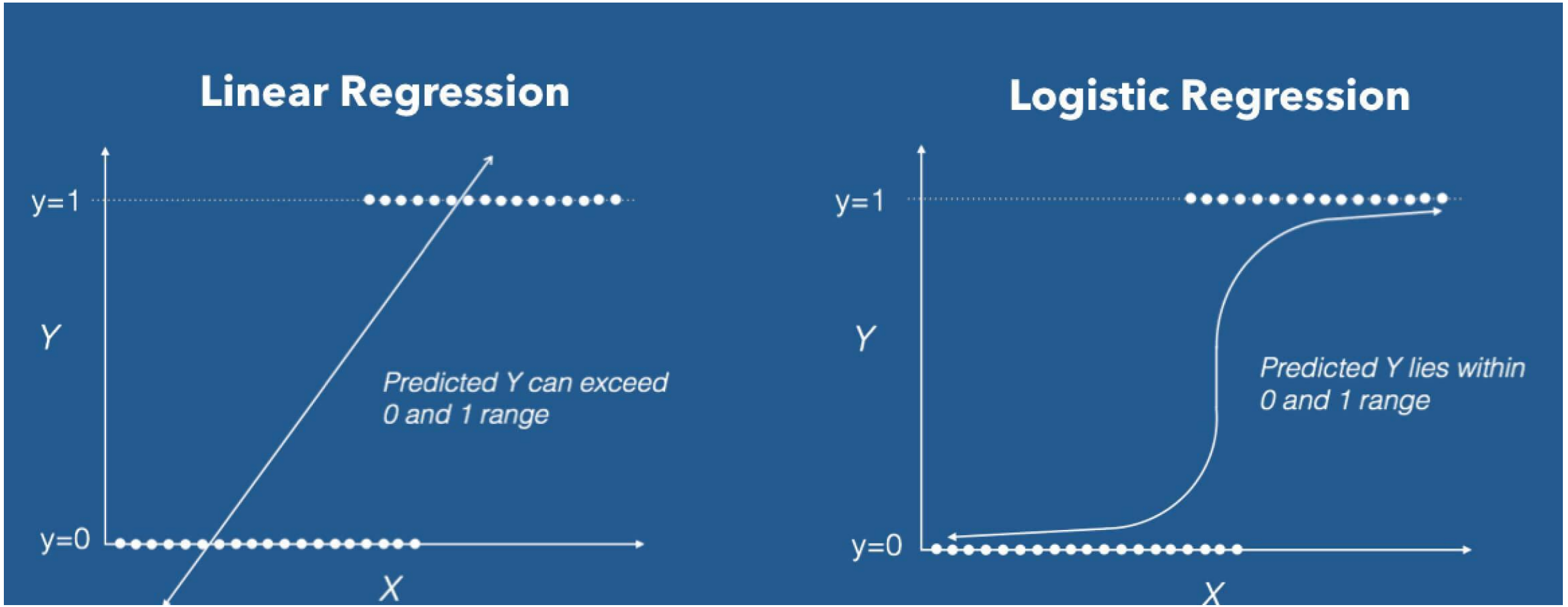
INTRODUCTION TO MACHINE LEARNING

DATA 602 Lecture 6

Dr. Tony Diana

tonydian@umbc.edu





Part 1
LOGISTIC REGRESSION

Logistic Regression

What is Logistic Regression?

- **Logistic Regression** is used when the **dependent variable** (target) is **categorical**
- Examples:
  - To predict whether an email is spam (1) or (0)
  - Whether the tumor is malignant (True) or not (False)
- Because **linear regression** is **unbounded**, it is not suitable for **classification problem**
- Using a **linear regression** would require setting a **threshold**
  - If the predicted outcome is 0.4 and the threshold value for a malignant tumor is 0.5, the predicted data point will be classified as 'non-malignant', which may have some serious consequences
- **Linear regression** works better on a **continuum of numeric estimates** and not on **class membership**
- **Logistic Regression** transforms linear regression's numeric estimates into a problem more apt of describing how a class fits an observation

Types of Logistic Regression

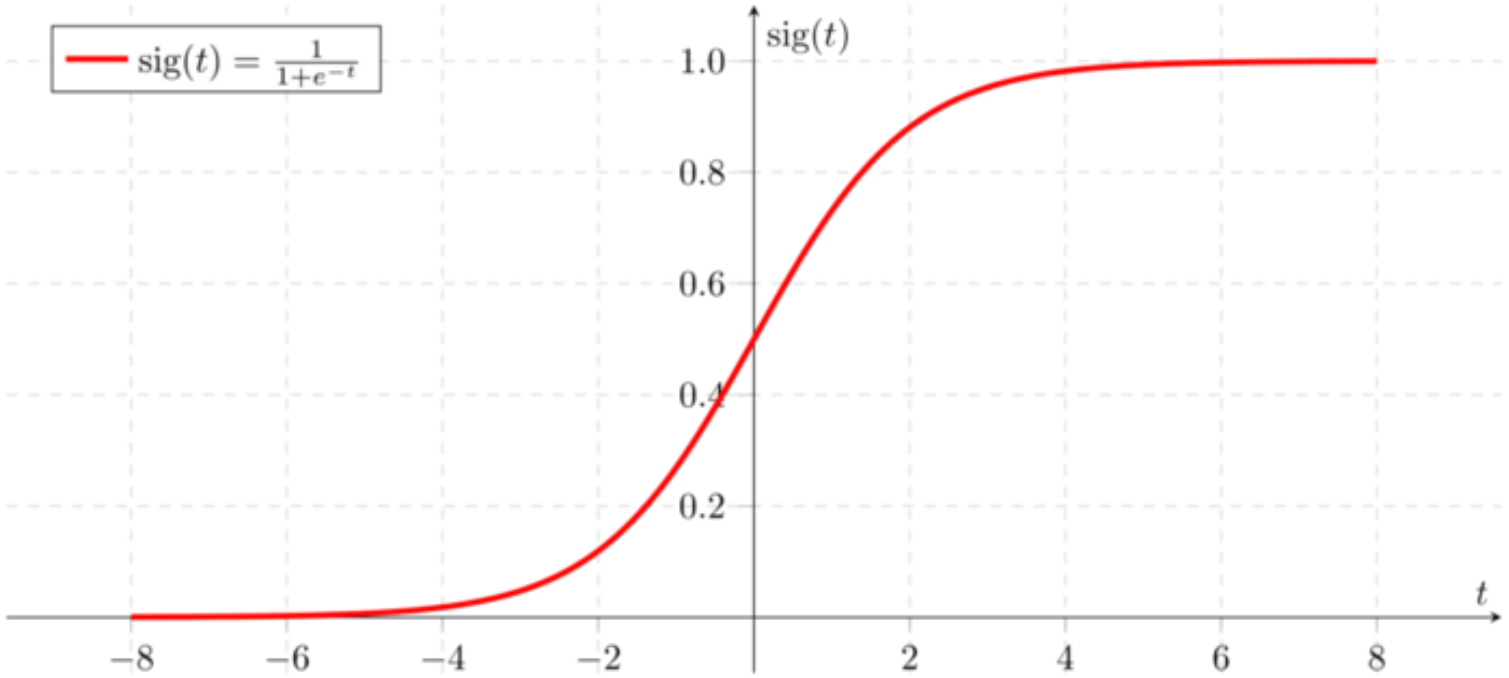
- **Binary Logistic Regression**: When the categorical response has only **two possible outcomes**
- **Multinomial Logistic Regression**: When there are **three or more categories without ordering**
- **Ordinal Logistic Regression**: When there are **three or more categories with ordering**

Logistic Regression can be used to make **predictions** (from sklearn.linear_model import LogisticRegression) or **classification** (from sklearn.linear_model import LogisticRegressionCV)

Key Assumptions

- Binary logistic regression requires the dependent variable to be 0 or 1
- For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome
- Only the meaningful variables should be included
- The independent variables should be independent of each other. That is, the model should have little or no multicollinearity
- The independent variables are linearly related to the **log odds**
- Usually, logistic regression requires quite large sample sizes

Representation of the Logistic Function



Model

Output = 0 or 1

Hypothesis: Z = WX + B

hΘ(x) = sigmoid (Z)

- If 'Z' goes to infinity, y(predicted) will become 1
- If 'Z' goes to negative infinity, y(predicted) will become 0
- $P(y^{(i)}=1) = \frac{1}{1 + \exp(-(\beta_0+\beta_1 x^{(i)1+\ldots+}\beta p x^{(i)}_p))}$

Regression Model (Multinomial)

```
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
X, y = load_iris(return_X_y=True)
clf = LogisticRegression(random_state=0, solver='lbfgs',
                    multi_class='multinomial').fit(X, y)
clf.predict(X[:2, :])

clf.predict_proba(X[:2, :])


clf.score(X, y)
print("Coefficients: ", clf.coef_)
print("Intercept: ", clf.intercept_)
print("Prediction: ", clf.predict(X))

Out:
Coefficients:  [[-0.41878528  0.96703041 -2.5209973  -1.08417682]
 [ 0.53124457 -0.31475282 -0.20008433 -0.94861142]
 [-0.1124593  -0.65227759  2.72108162  2.03278825]]
Intercept:  [  9.84037025   2.21700109 -12.05737134]
Prediction:  [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1
 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Classification Model (Multinomial)

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegressionCV
X, y = load_iris(return_X_y=True)
clf = LogisticRegressionCV(cv=5, random_state=0,
                    multi_class='multinomial').fit(X, y)
clf.predict(X[:2, :])

clf.predict_proba(X[:2, :]).shape

Print(“Accuracy Score:\n”, clf.score(X, y))

Out:
Accuracy Score:
0.98
```

UMBC

The Cost Function

$$\text{Cost}(h_\Theta(x), Y(\text{actual})) = -\log(h_\Theta(x)) \text{ if } y=1$$

$$-\log(1- h_\Theta(x)) \text{ if } y=0$$

- Linear regression uses **Mean Squared Error (MSE)** as its **cost function**
- If **MSE** is used for logistic regression, then it will be a non-convex function of parameters (θ or theta) Gradient descent will converge into global minimum only if the function is convex
- Instead of Mean Squared Error, we use a cost function called **Cross-Entropy**, also known as **Log Loss**
- **Cross-entropy loss** or **log loss** measures the performance of a classification model whose output is a probability value between 0 and 1
- **Cross-entropy loss** increases as the predicted probability diverges from the actual label
  - So predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0
- **Cross-entropy loss** can be divided into two separate cost functions: one for y = 1 and one for y = 0

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

UMBC

Interpretation of the Logistic Regression Model Coefficients

$$\log(P(y = 1) /1 - P(y = 1)) = \log(P(y = 1) / P(y = 0)) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$$

- We call the term in the log() function "**odds**" (probability of event divided by probability of no event)
- When wrapped in the logarithm, it is called **log odds**

- **If y = 1 , a change in a feature by one unit changes the odds ratio (multiplicative) by a factor of exp(β_j)**
- In other words, if y = 1, a change in x_j by one unit increases the log odds ratio by the value of the corresponding weight
- If you have odds of 2, it means that the probability for y = 1 is twice as high as y = 0
- If you have a weight (= log-odds ratio) of 0.7, then increasing the respective feature by one unit multiplies the odds by exp(0.7) (that is, approximately 2) and the odds change to 4

UMBC

Logistic Regression: The C Hyperparameter

- **C** represents the trade-off parameter of the logistic regression which determines the strength of the regularization
- High values of **C** imply less regularization and vice versa
- **C** is the inverse of regularization strength or lambda
- **C** provides an indication of the tolerance for misclassification
- If **C** is low, then there is a potential for bias
- If **C** is high, then it means that we allow more misclassifications that can result in greater variance
- **Grid Search** is the best way to finding the optimal **C** value:
  - f**rom sklearn.model_selection import GridSearchCV**
- C is a **regularization factor** that affects the classification of the data

Logistic Regression Outputs: Example of Interpretation

We use the logistic regression model to predict cervical cancer based on some risk factors. The following table shows the estimate weights, the associated odds ratios, and the standard error of the estimates.

TABLE 4.1: The results of fitting a logistic regression model on the cervical cancer dataset. Shown are the features used in the model, their estimated weights and corresponding odds ratios, and the standard errors of the estimated weights.

	Weight	Odds ratio	Std. Error
Intercept	2.91	18.36	0.32
Hormonal contraceptives y/n	0.12	1.12	0.30
Smokes y/n	-0.26	0.77	0.37
Num. of pregnancies	-0.04	0.96	0.10
Num. of diagnosed STDs	-0.82	0.44	0.33
Intrauterine device y/n	-0.62	0.54	0.40

Exp(-0.82)

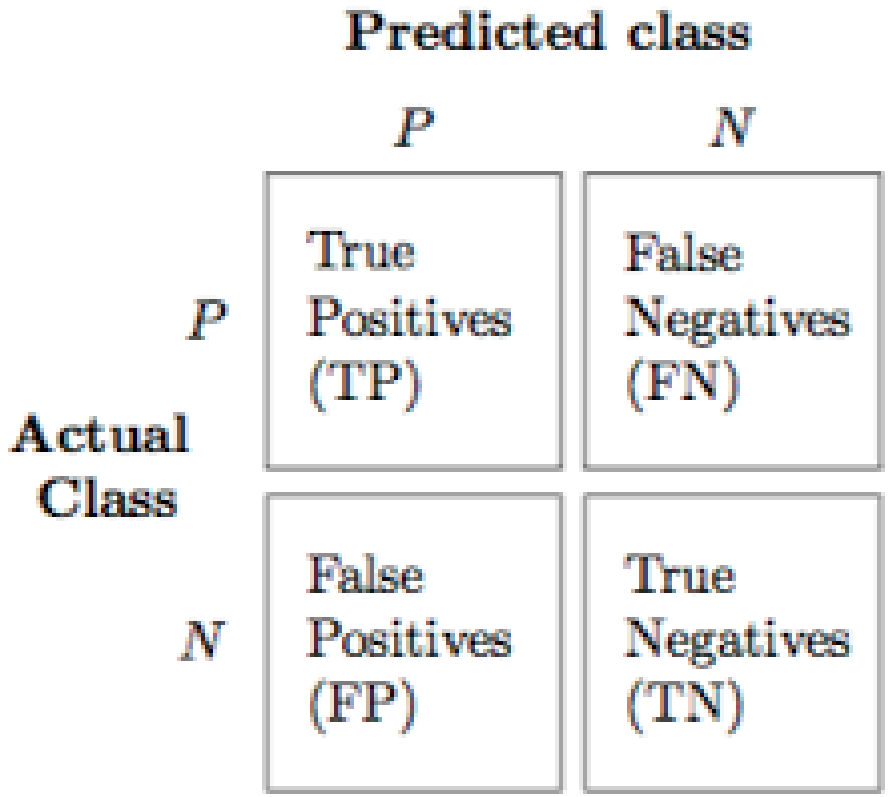
Interpretation of a numerical feature ("Num. of diagnosed STDs"): An increase in the number of diagnosed STDs (sexually transmitted diseases) changes (decreases) the odds of cancer vs. no cancer by a factor of 0.44, when all other features remain the same. Keep in mind that correlation does not imply causation. No recommendation here to get STDs.

Interpretation of a categorical feature ("Hormonal contraceptives y/n"): For women using hormonal contraceptives, the odds for cancer vs. no cancer are by a factor of 1.12 higher, compared to women without hormonal contraceptives, given all other features stay the same.

Like in the linear model, the interpretations always come with the clause that 'all other features stay the same'.

Source: https://christophm.github.io/interpretable-ml-book/logistic.html

The Confusion Matrix



TPR (Sensitivity) = TP /TP + FN
TNR (Specificity) = TN / TN + FP
FPR = 1- Specificity

A **confusion matrix** is a table that is often used to **describe the performance of a classification model** (or "classifier") on a set of test data for which the true values are known

- **True positives (TP):** These are cases in which we predicted yes (they have the disease), and they do have the disease
- **True negatives (TN):** We predicted no, and they do not have the disease
- **False positives (FP):** We predicted yes, but they do not actually have the disease. (Also known as a "Type I error")
- **False negatives (FN):** We predicted no, but they actually do have the disease. (Also known as a "Type II error")

The Classification Report

```
from sklearn import metrics
import numpy as np
y_pred = np.around(model.predict(x_test))
print(metrics.classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.92	0.93	0.92	12500
1	0.93	0.92	0.92	12500
micro avg	0.92	0.92	0.92	25000
macro avg	0.92	0.92	0.92	25000
weighted avg	0.92	0.92	0.92	25000

Accuracy = (TP + TN) / (TP + FN + FP + TN)

Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

F1-Score = 2*[(Precision * Recall) / (Precision + Recall)]

Specificity = TN / (TN + FP)

UMBC

The Performance Metrics of a Logistic Regression Classification Model

There are several metrics used to assess the performance of the classification model:

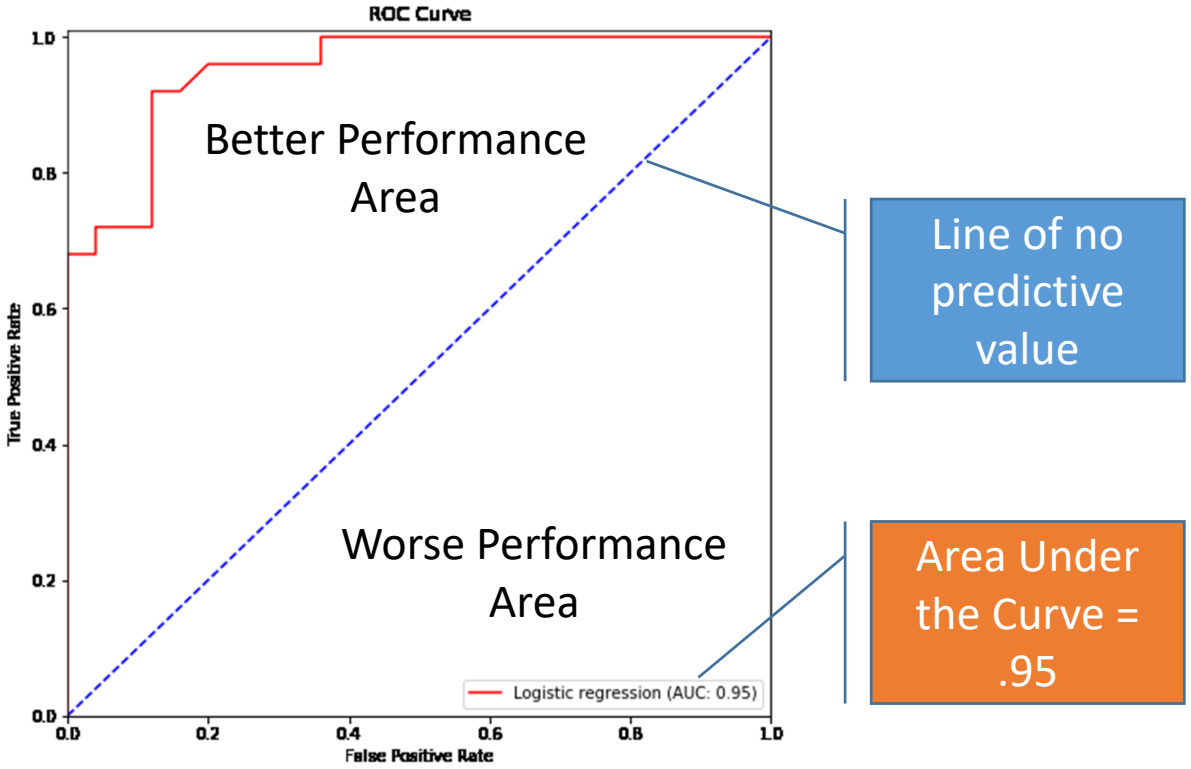
- **Accuracy**: the percentage of outcomes that were predicted with the correct tag (for instance, 'negative,' 'positive,' or 'neutral,' or 'yes' or 'no'). **It is the fraction of predictions the model got right**
- **Precision**: the percentage of outcomes a classifier got right out of the total number of outcomes that a logistic regression model predicted for a given tag. **It is the fraction of relevant instances**
- **Recall (sensitivity)**: the percentage of outcomes a classifier predicted for a given tag out of the total number of outcomes it should have predicted for that given tag. **It is the fraction of total relevant results correctly classified by the algorithm**
- **F1 score**: the harmonic mean of precision and recall
- **Specificity**: the proportion of TN instances predicted to be negative

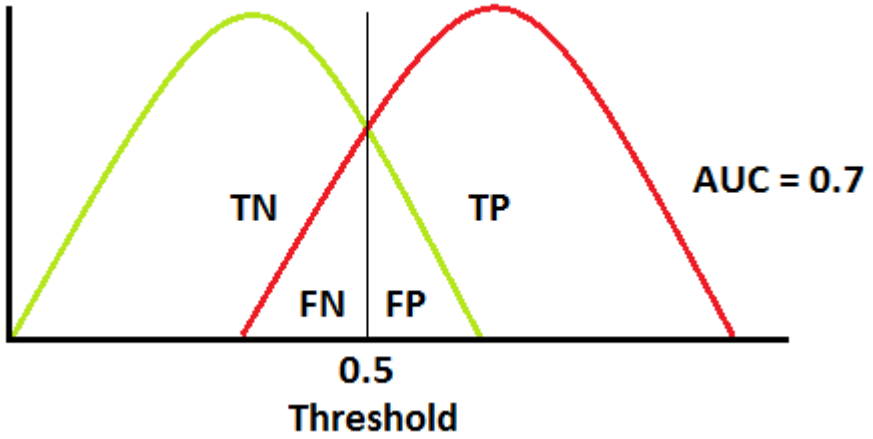
Interpretation of Specificity and Sensitivity

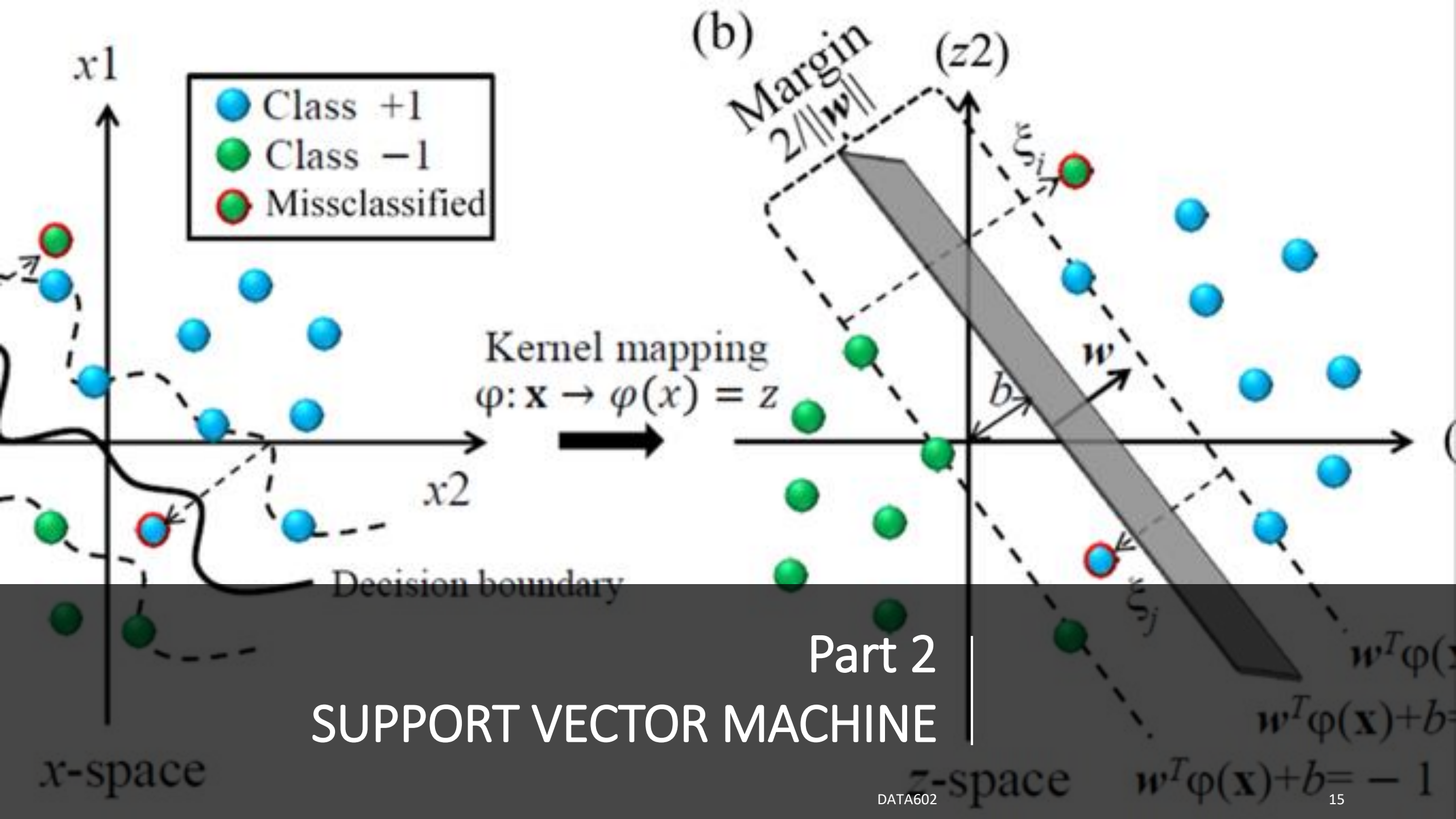
- **Specificity** → 98% specificity means that 98% of the patients who truly do not have the disease were predicted not to have it
- **Sensitivity** → 98% sensitivity or recall means that 98% of the patients who truly have the disease were predicted to have it

The Receiver Operating Curve (ROC)

- It is a commonly used graph that summarizes the performance of a classifier over all possible thresholds
- It is generated by plotting the True Positive Rate (y-axis) against the False Positive Rate (x-axis) as you vary the threshold for assigning observations to a given class
- ROC is a **probability curve** and the **Area Under the Curve** (AUC) represent the **degree** or **measure of separability**. It tells us how **much the model is capable of distinguishing between classes**





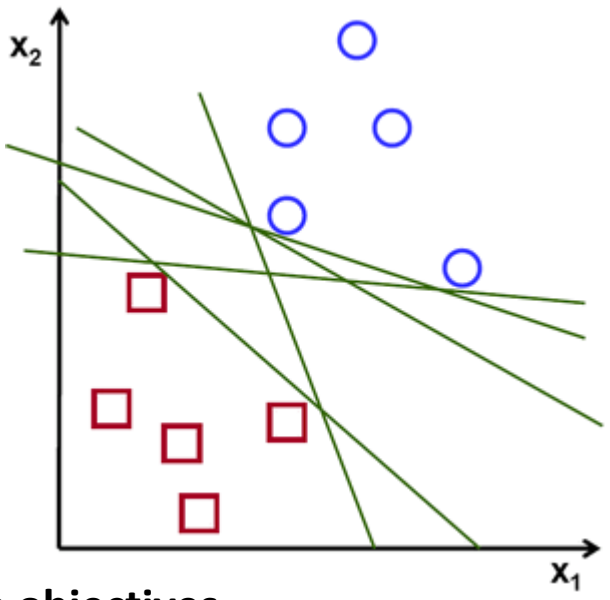


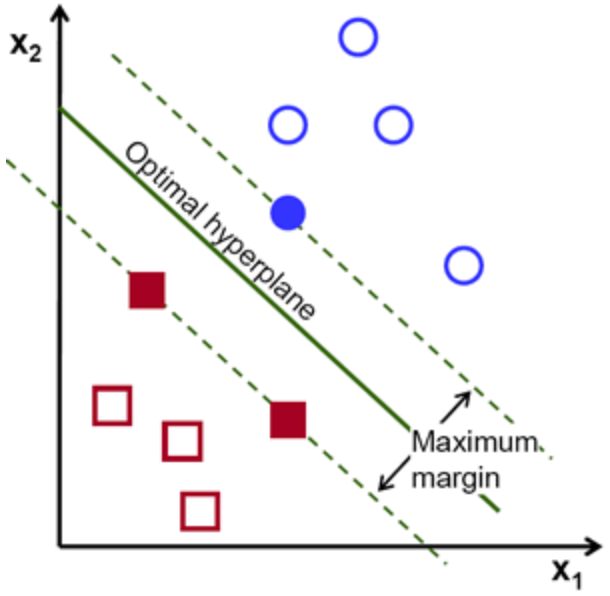
Part 2
SUPPORT VECTOR MACHINE

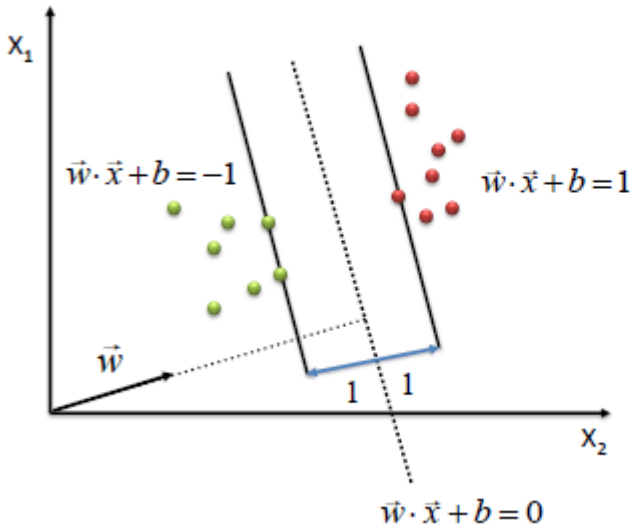
Support Vector Machine

What is Support Vector Machine (SVM)?

- It is used for both regression and classification tasks. Vapnick and his colleagues at AT&T Bell Laboratories developed SVM in the 1990s
- The objective of the SVM algorithm is to find a hyperplane in an N-dimensional space (N being the number of features) that distinctly classifies the data points
- **Hyperplanes** are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different **classes**







$$\max \frac{2}{\|w\|}$$

s.t.

$(w\cdot x+b)\geq 1, \forall x$ of class 1

$(w\cdot x+b)\leq -1, \forall x$ of class 2

The objectives

- To find **a plane that has the maximum margin**, i.e. the maximum distance between data points of both classes
- All the data points that fall on one side of the line will be labeled as one class and all the points that fall on the other side will be labeled as the second

The Benefits and Disadvantages of SVM

Benefits

- The SVM model depends on few support vectors → model is very compact
- It provides fast predictions
  - The equation for making a prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

$$f(x) = \beta_0 + \sum \alpha_i (i * (x, x_i))$$

  - The coefficients β_0 and α_i (for each input) must be estimated from the training data by the learning algorithm
- It works well with high-dimensional data because of the dependence on points near the margin
- The integration with **kernel** methods (kernel trick) makes them very versatile and adapt to different type of data
- Kernel methods convert not separable problem to separable problem. Here are some types:
  - **Linear Kernel SVM**: $K(x, x_i) = \sum (x * xi)$
  - **Polynomial Kernel SVM**: $K(x, x_i) = 1 + \sum (x * xi)^2$
  - **Radial Kernel SVM**: $K(x, x_i) = e^{-\gamma * \sum (x - xi^2)}$

Gamma defines how far the influence of a single training example reaches, with low value meaning 'far' and high value meaning 'close'

Disadvantages

- The SVM model does not work well with large datasets: training takes longer
- It is sensitive to noisy data: target classes may be overlapping
- It does not provide probability estimates

SVM/The Loss Function

$$L = \frac{1}{2}\|\vec{w}\|^2 - \sum_i^n \alpha_i \left[y_i(\vec{w}\cdot\vec{x}+b)-1\right] \qquad \vec{w} = \sum_i^n \alpha_i y_i x_i \quad (1)$$

$$L = \frac{1}{2}\left(\sum_i^n \alpha_i y_i \vec{x}_i\right)\cdot\left(\sum_i^n \alpha_j y_j \vec{x}_j\right) - \sum_i^n \alpha_i y_i \vec{w}\cdot\vec{x} - \sum_i^n \alpha_i y_i b + \sum_i^n \alpha_i$$

$$\vec{w} = \sum_i^n \alpha_i y_i x_i \quad (1) \qquad \sum_i^n \alpha_i y_i = 0 \quad (2)$$

$$L = \frac{1}{2}\sum_i^n\sum_j^n \alpha_j y_j \alpha_i y_i \vec{x}_i\cdot\vec{x}_j - \sum_i^n\sum_j^n \alpha_j y_j \alpha_i y_i \vec{x}_i\cdot\vec{x}_j - 0 + \sum_i^n \alpha_i$$

$$L = \sum_i^n \alpha_i - \frac{1}{2}\sum_i^n\sum_j^n \alpha_i\alpha_j y_i y_j x_i\cdot x_j$$

Source: Toward Data Science

SVM: The C, Gamma, and Epsilon Parameters

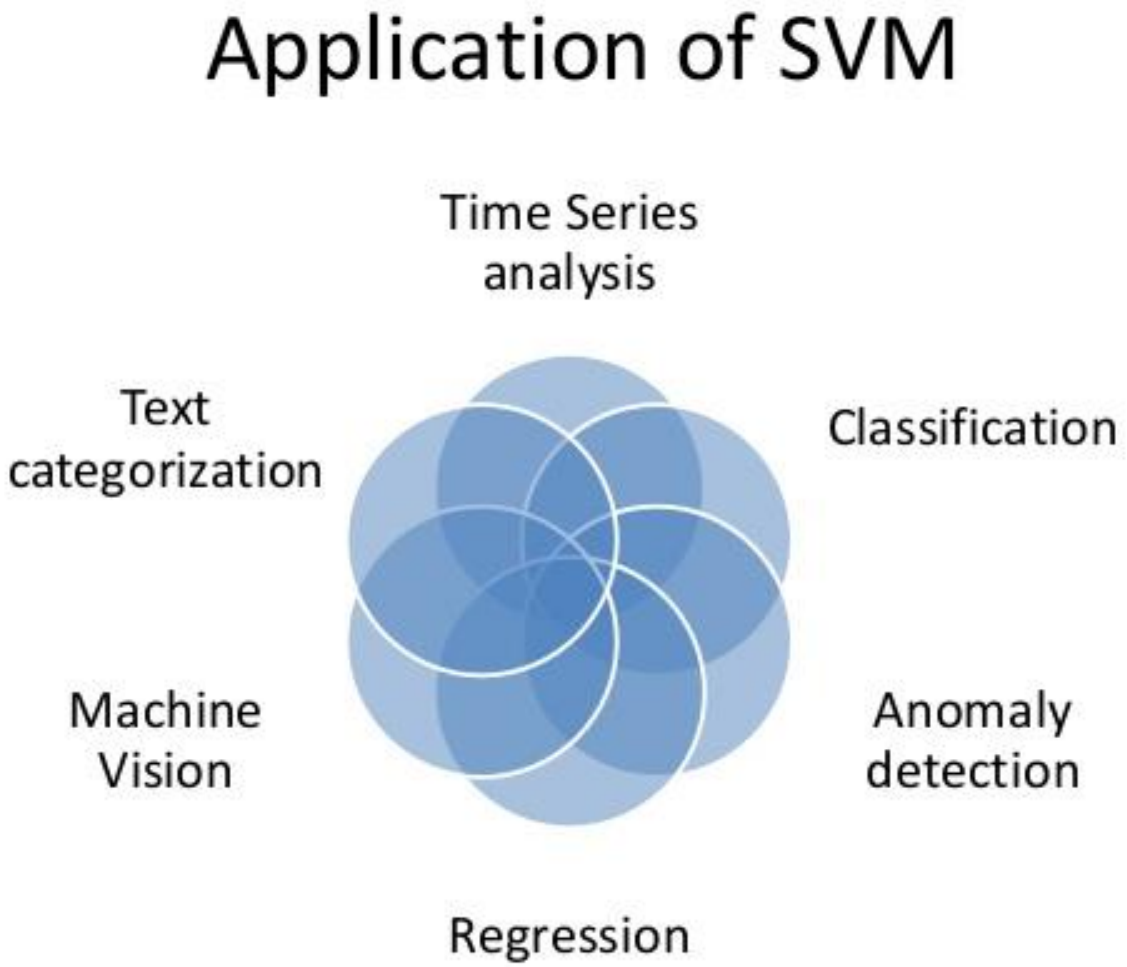
- **C** indicates the number of mismatches that the SVM model can ignore when learning
- If we have too much correction (a large **C** value), the search will accept too many exceptions, which creates **overfitting**
- If the value of **C** is too low, the SVM will look for a complete separation of points. As a result, we get a suboptimal hyperplane whose margin is too small
- **C** represents the **cost of mismatch**
  - Higher **C** value→ narrower margin → higher costs in case of misclassification (higher penalty) → **higher bias**
  - Lower **C** value → wider margin → more integration of wrongly predicted outcomes (lower penalty) → **higher variance**
- **Epsilon** creates exceptions in classification and allows a more realistic fit of the SVM formulation when data are noisy
- **Epsilon** affects the complexity and generalization capability of the model
- The value of **epsilon** can affect the number of support vectors used to construct the regression function
  - The bigger ε, the fewer support vectors are selected
  - Bigger ε-values result in more flat estimates

SVM: The C, Gamma, and Epsilon Parameters (Continued)

- **Gamma** represents the radius of the points (or 'bubbles') or the shape parameter for the **Radial-Based Function (RBF) and Gaussian**
- **Gamma** determines the outreach of the support vectors
- Scikit-Learn's current default setting for gamma is 'auto'
  - In earlier version, the value of gamma was 1 / n_features. In later versions of sklearn, the default is 'scale,' which takes the value 1 / (n_features * X.var())
  - If gamma is too large, the radius of the area of influence of the support vectors only includes the support vector itself and no amount of regularization with C will be able to prevent overfitting. Larger values decrease the effect of any individual support vector
  - When gamma is very small, the model is too constrained and cannot capture the complexity or "shape" of the data. The region of influence of any selected support vector would include the whole training set. The resulting model will behave similarly to a linear model with a set of hyperplanes that separate the centers of high density of any pair of two classes

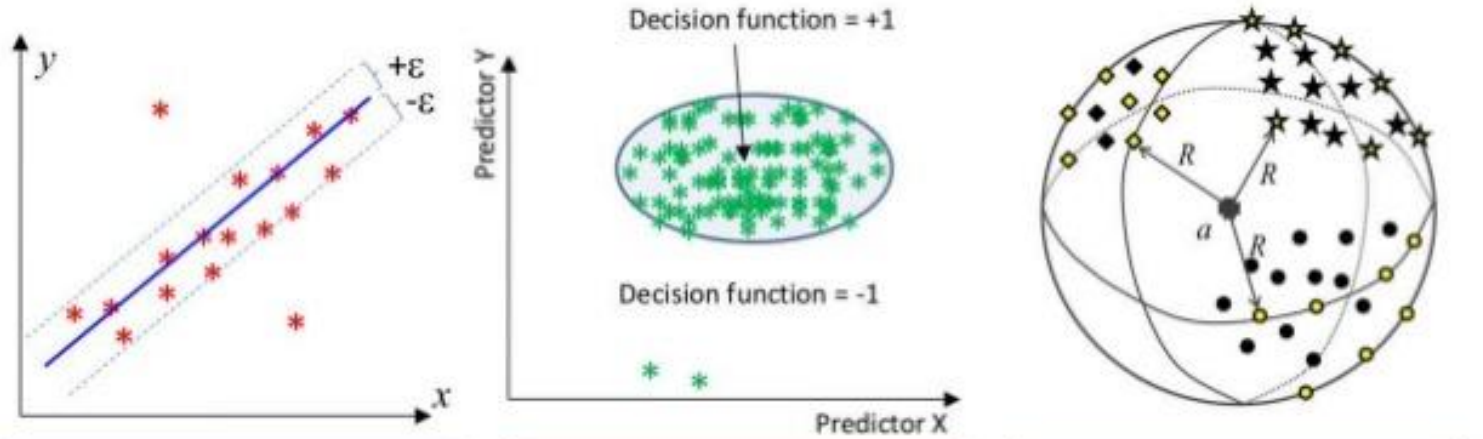
Applications of SVM

Digg Data



Digg Data

SVM usage beyond classification



Regression analysis (ε-Support vector regression)

Anomaly detection (One-class SVM)

Clustering analysis (Support Vector Domain Description)

Likelihood

Class Prior Probability

$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

Posterior Probability

Predictor Prior Probability

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

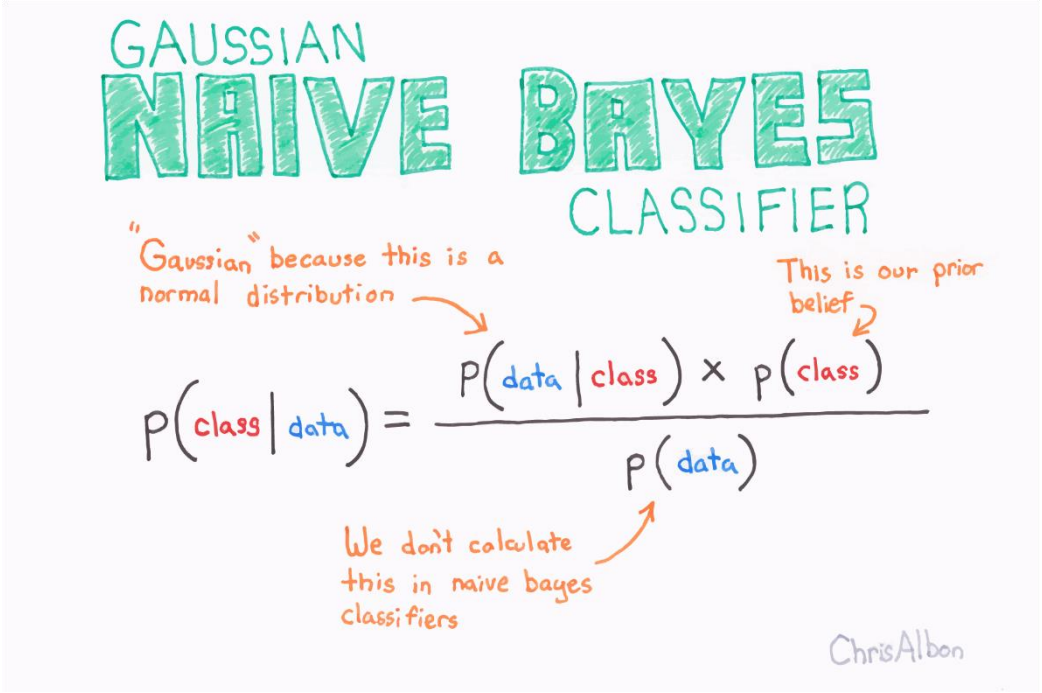
Part 3
NAÏVE BAYES

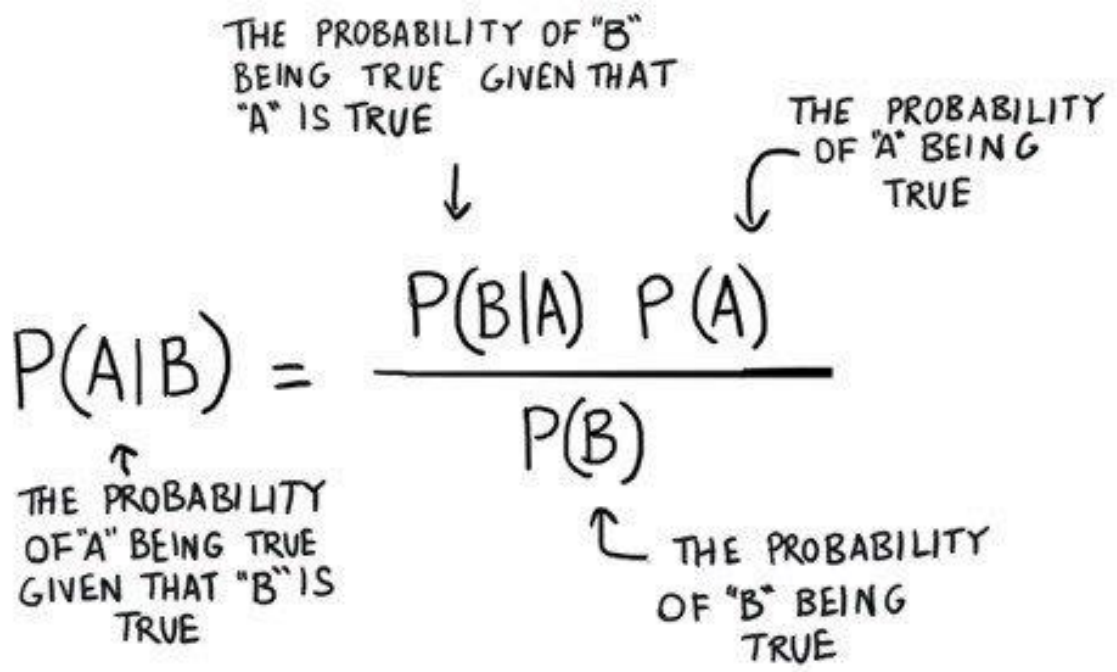
Naïve Bayes Classifier

What is Naïve Bayes Classification?

- It is a classification algorithm based on the Naïve Bayes theorem
- The algorithm assumes independence among predictors
- Naive comes the fact that the presence of a particular feature in a class is assumed to be unrelated to the presence of any other feature
  - A fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter
  - Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'
- Naïve Bayes models work well with large data sets. The Naïve Bayes algorithm is known for outperforming more sophisticated classification methods
- Bayes theorem provides a way of calculating posterior probability $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$ with
  - P(A|B) as the posterior probability of class (A, target) given predictor(B, attributes)
  - P(B|A) as the likelihood which is the probability of predictor given class
  - P(A) as the class prior probability of class, and
  - P(B) as the predictor prior probability of predictor

Naïve Bayes Classifier





$$\text{Posterior} = \frac{\text{Likelihood} * \text{Prior}}{\text{Evidence}}$$

$$P(target|\ predictor) = \frac{P(predictor\ |\ target)\ .\ P(target)}{P(predictor)}$$

Naïve Bayes Classifier

The Pros

- It is easy to predict class of test dataset
- It performs well in multi-class prediction
- When the assumption of independence holds, a NB classifier performs better than other algorithms such as LR and fewer training data are needed
- It performs well with categorical input variables compared to numerical variables. For numerical variable, normal distribution is assumed

The Cons

- If a categorical variable has a category in the test dataset, which is not observed in train dataset, the model will assign a zero probability and will be unable to make a prediction. This is referred to as ‘zero frequency’. Laplace estimation as a smoothing technique can be used to solve this
- It is not always performing well as an estimator. One needs to be careful about the outputs from ‘predict_proba’
- In real life, it is not always possible to have independence among a set of predictors

The Types of Naïve Bayes Algorithms

- **Gaussian:** it assumes that features follow a normal distribution
- **Multinomial:** It is used for discrete counts. It is used in text classification. We can think of the number of times outcome number *x* is observed in *n* trials
- **Bernoulli:** NB is useful if the feature vectors are binary (zeros and ones). One application in text classification is the 'bag of words' model where the 1's and 0's are words occurring or not in a document

Applications

- **Real Time Prediction**
- **Multi-Class Prediction**
- **Text Classification/Spam Filtering/Sentiment Analysis:** NB is mostly used in text classification (due to better result in multi-class problems and independence rule). It performs better than other algorithms
- **Recommendation System:** NB classifier and collaborative filtering together build a recommendation system that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a product/service or not