

GCT634/AI613: Musical Applications of Machine Learning

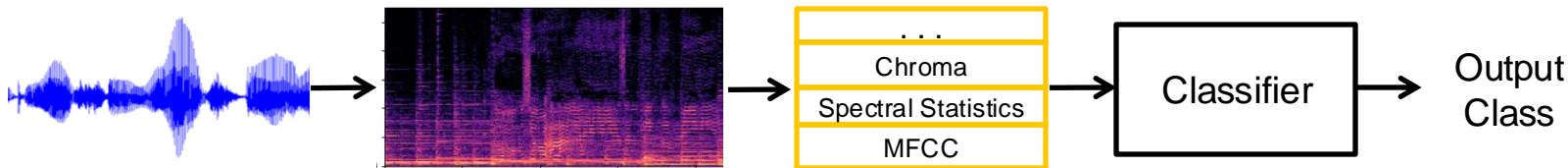
Music Classification: Traditional Machine Learning



Juhan Nam

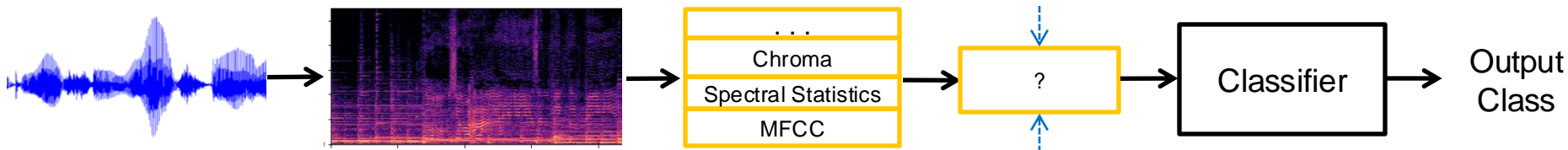
Introduction

- Traditional machine learning pipeline in classification tasks
 - Select a set of audio features and concatenate for a given task
 - Frame-wise differences (i.e., delta and double-delta) are often concatenated to capture local temporal changes
 - The concatenated features are complementary to each other



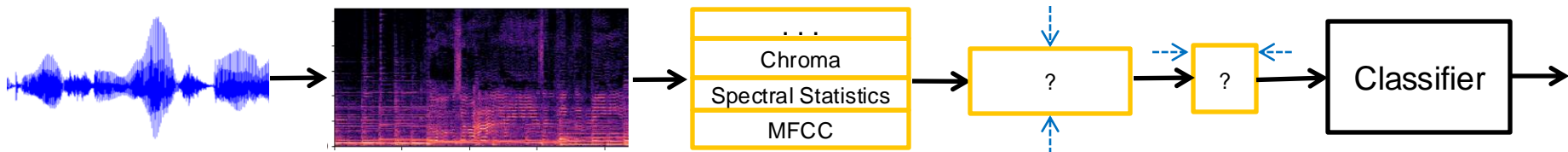
Issues: Redundancy and Dimensionality

- Redundancy and dimensionality in the concatenated audio features
 - Adding more features increases the dimensionality of the feature vectors. As a result, the classifier needs more parameters to train
 - The information in the concatenated feature vectors can be redundant



Issues: Temporal Summary

- Taking the entire frames of audio features over time is too large
 - 1 to 5 seconds of audio is a typical size as the input of classifier
 - The size is often called **texture window** or **context window**
- Summary methods
 - Temporal pooling (average, standard deviation): orderless summary
 - DCT over time: take a small number of low-frequency cosine kernels for each feature dimension (1D DCT) or the entire feature dimension (2D DCT)



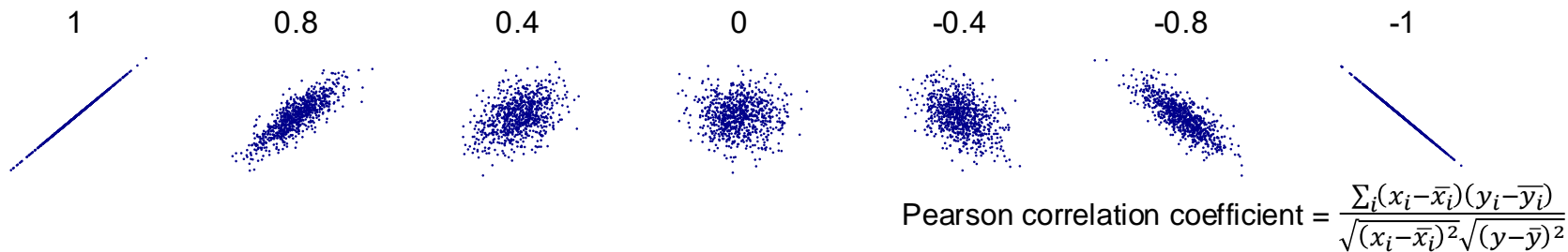
Unsupervised Learning

- Dimensionality reduction
 - Project data from a high-dimensional space to a low-dimensional space
 - Find correlation or redundancy patterns between data elements
 - Examples: PCA, NMF, Auto-Encoder, t-SNE, UMAP
- Clustering
 - Discover groups of similar examples with the data
 - Examples: K-means, GMM, Spectral clustering, hierarchical clustering
- Density estimation
 - Fit a probabilistic distribution model to data
 - The model can explain the likelihood of samples
 - Examples: GMM, Kernel density estimation

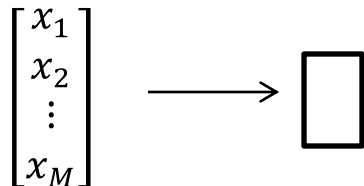
Principal Component Analysis

- Correlation and Redundancy

- We can measure the redundancy between two elements in a feature vector by computing their correlation

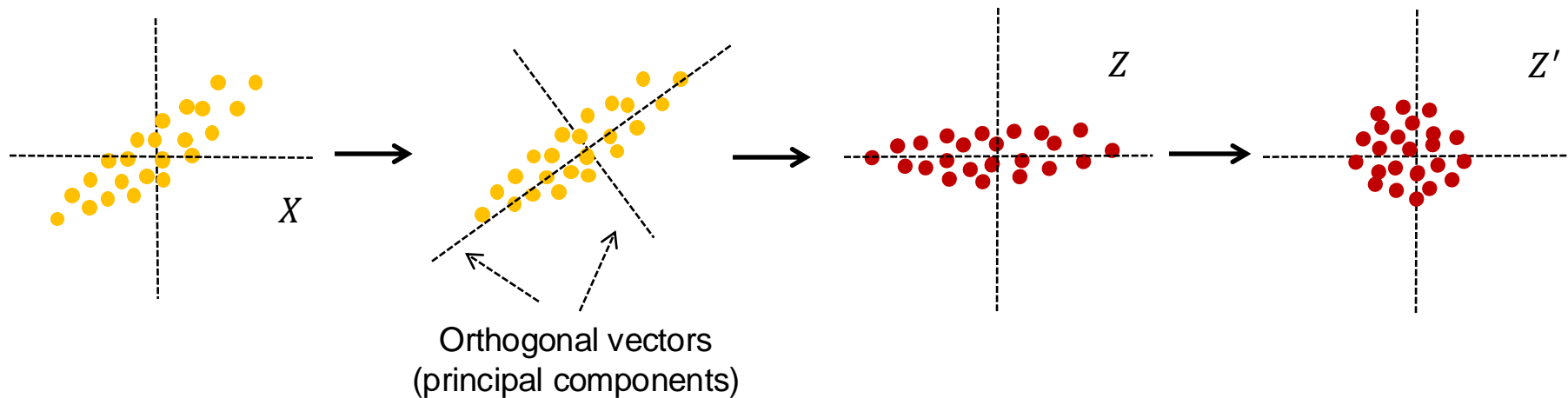


- If some of the elements have high correlations, we can remove the redundant elements



Principal Component Analysis

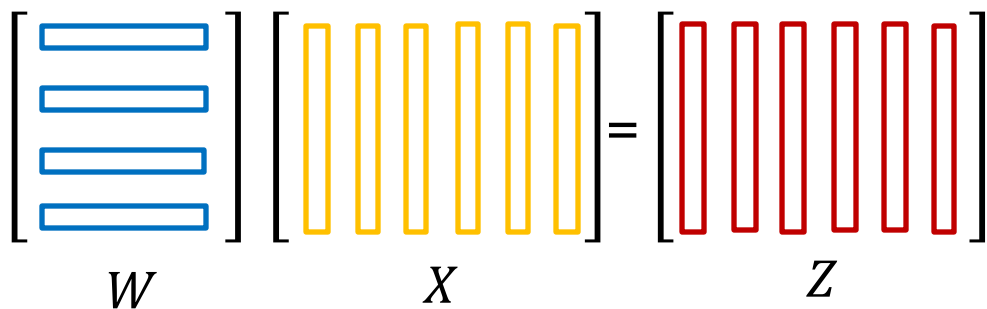
- Transform the input space (X) into a latent space (Z) such that the latent space is de-correlated (i.e., each dimension is orthogonal to each other)
 - Linear transform designed to maximize the variance of the first principal component and minimize the variance of the last principle component



Principal Component Analysis

- Transform the input space (X) into a latent space (Z) such that the latent space is de-correlated (i.e., each dimension is orthogonal to each other)
 - Linear transform designed to maximize the variance of the first principal component and minimize the variance of the last principle component

$$WX = Z$$



$$ZZ^T = N = \begin{bmatrix} n_1 & 0 & 0 & 0 \\ 0 & n_2 & 0 & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & n_M \end{bmatrix}$$

The diagonal elements correspond to the variances of transformed data points on each dimension

Principal Component Analysis: Eigenvalue Decomposition

- Eigenvalue decomposition (Q : eigenvectors, Λ : eigenvalue matrix)

$$Ax_i = \lambda_i x_i \longrightarrow AQ = Q\Lambda \longrightarrow A = Q\Lambda Q^{-1} \longrightarrow \underline{A = Q\Lambda Q^T}$$

$Q = [x_1 x_2 \dots x_N]$ $\Lambda = \text{diag}(\lambda_i)$ (If A is symmetric)

- To derive W

$$ZZ^T = N \longrightarrow (WX)(WX)^T = N \longrightarrow WXX^T W^T = N \longrightarrow W\text{Cov}(X)W^T = N$$

$\longrightarrow \underline{\text{Cov}(X) = W^T N W}$ W is an orthogonal matrix ($W^{-1} = W^T$)

$$\text{Cov}(X) = XX^T = (XX^T)^T = \text{Cov}(X)^T \quad \text{Covariance matrix of } X \text{ is symmetric}$$

$$\boxed{W = Q^T} \quad N = \Lambda$$

The orthogonal matrix W is obtained from the eigenvectors of covariance matrix of X !

Principal Component Analysis: Eigenvalue Decomposition

- From the eigenvalue decomposition

$$A = Q\Lambda Q^T \rightarrow Q^T A Q = \Lambda \rightarrow Q^T A Q = \Lambda^{1/2} \Lambda^{1/2} \rightarrow (\Lambda^{-1/2} Q)^T A (\Lambda^{-1/2} Q) = I$$

- Set the scaled orthogonal matrix

$$W' = \Lambda^{-1/2} Q^T = \Lambda^{-1/2} W$$

W' becomes an orthonormal matrix

$$\Lambda^{-1/2} = \begin{bmatrix} 1/\sqrt{\lambda_1} & 0 & & \\ 0 & 1/\sqrt{\lambda_2} & & \\ & & \ddots & \\ & \dots & 0 & 1/\sqrt{\lambda_M} \end{bmatrix}$$

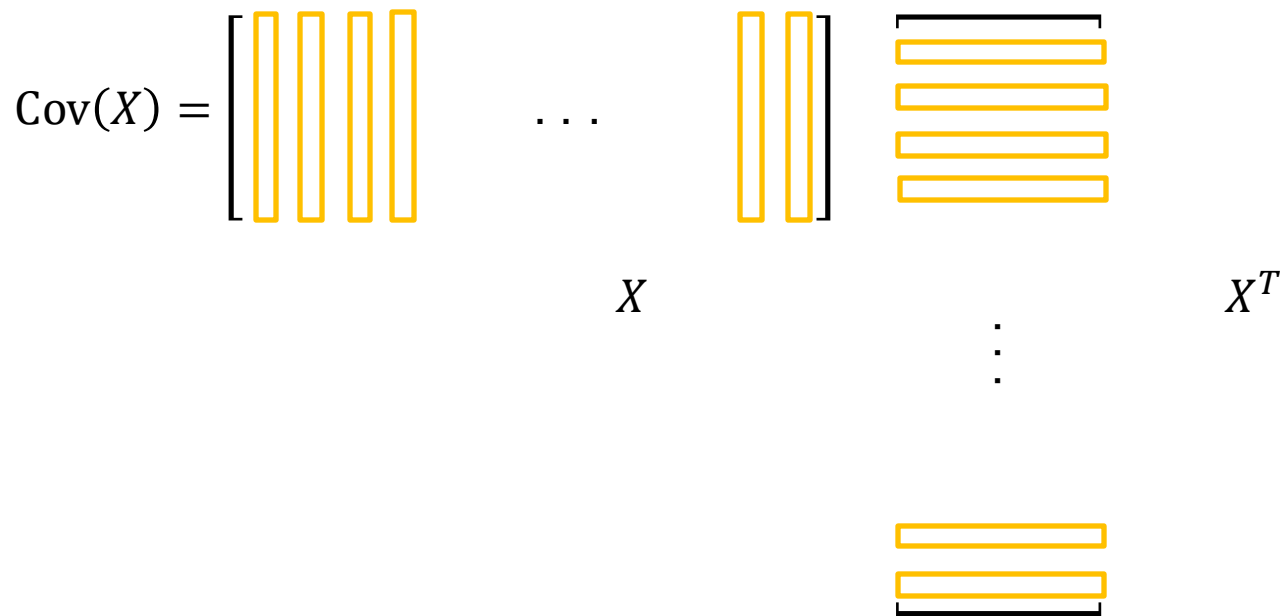
- The latent spaced is normalized to have unit variances

$$Z' Z'^T = (W' X)(W' X)^T = W' X X^T W'^T = \Lambda^{-1/2} (W X)(X^T W^T) \Lambda^{-1/2} = \Lambda^{-1/2} Z Z^T \Lambda^{-1/2} = I$$

$$Z' = \Lambda^{-1/2} Z$$

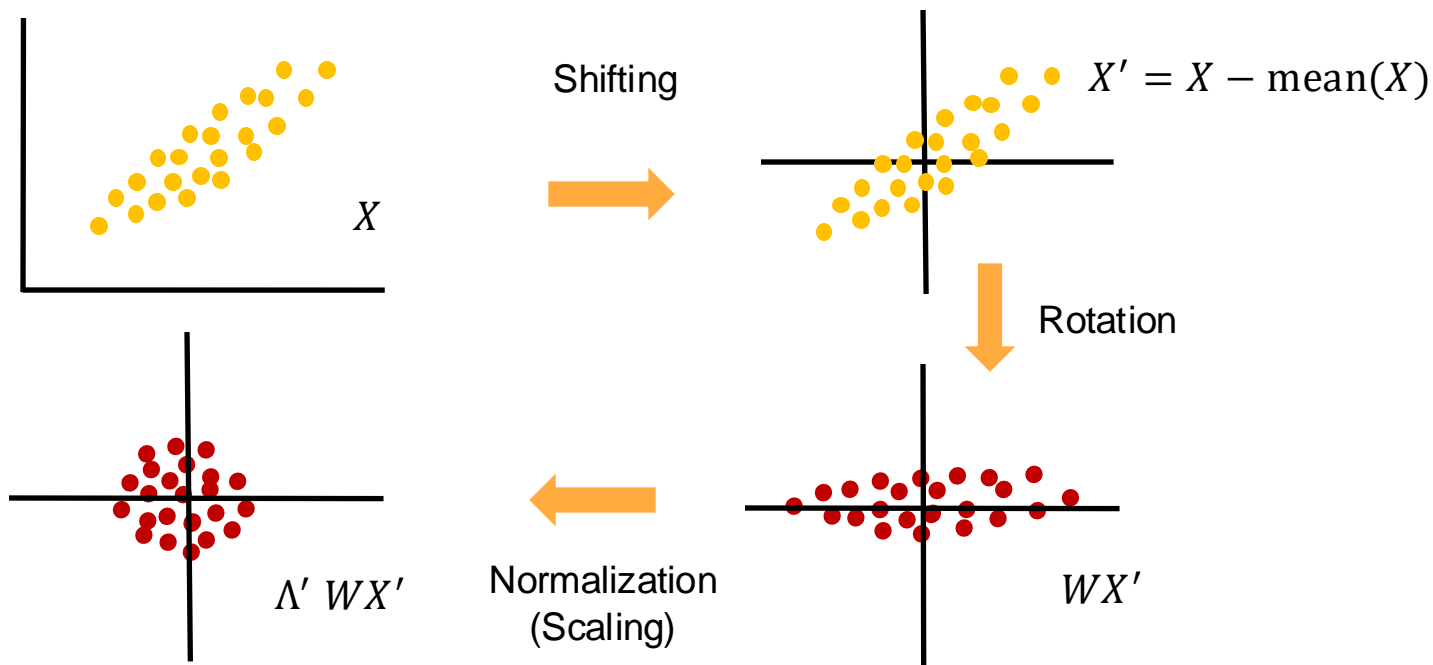
Principal Component Analysis in Practice

- X is a huge matrix where each column is a data point in practice
 - Computing the covariance matrix is a bottleneck
 - We often randomly sample the input data



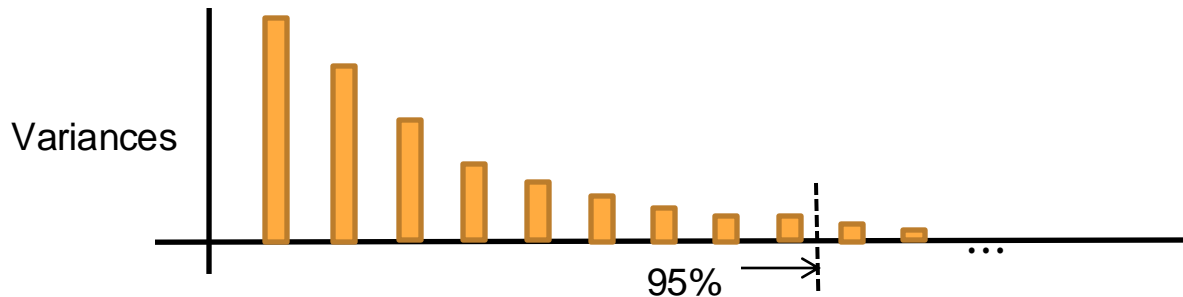
Principal Component Analysis in Practice

- Shift the distribution to have zero mean
- The normalization is optional: called PCA whitening



Dimensionality Reduction Using PCA

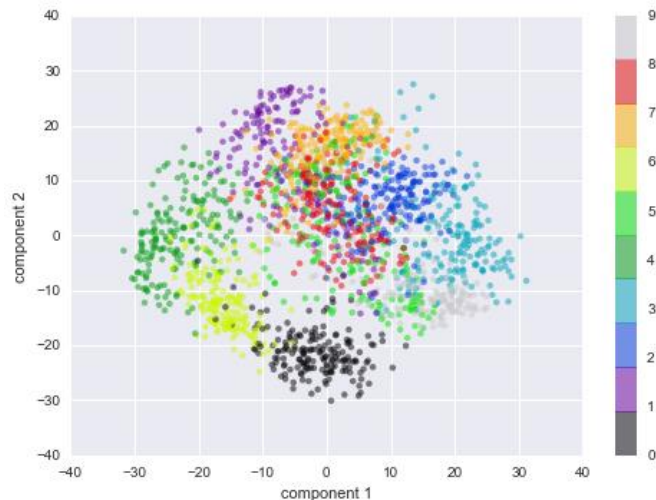
- We can remove principal components with small variances
 - Sort the variances in the latent space (the eigenvalues) in descending order and removing the tails
 - A strategy is accumulating the variances from the first principal component. When it reaches 90% or 95% of the sum of all variances, remove the remaining dimensions. This significantly reduces the dimensionality.



- Note that you can reconstruct the original data with some loss
 - You can use PCA as a data compression method

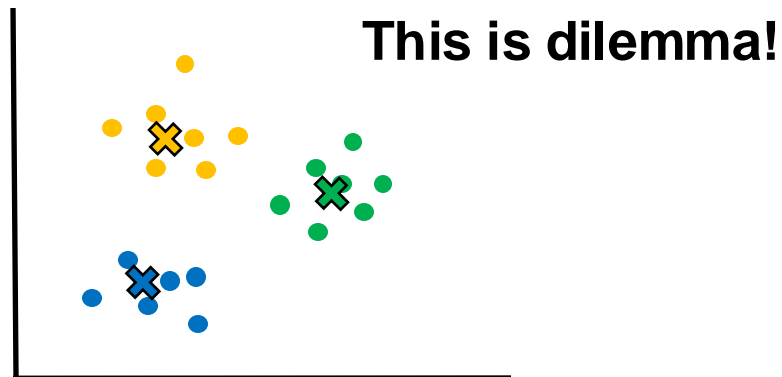
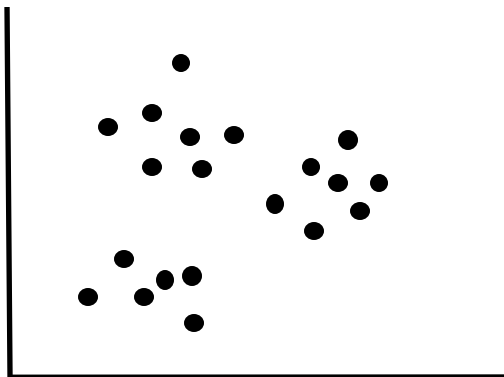
Visualization Using PCA

- Taking the first two or three principal components only for 2D or 3D visualization
 - A popularized used feature visualization method along with t -SNE in analyzing the latent feature space in the trained deep neural network



K-Means Clustering

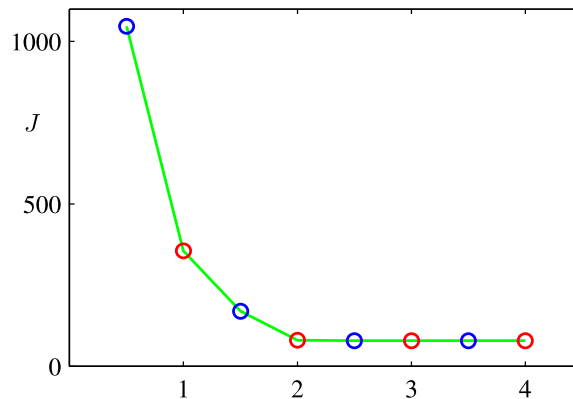
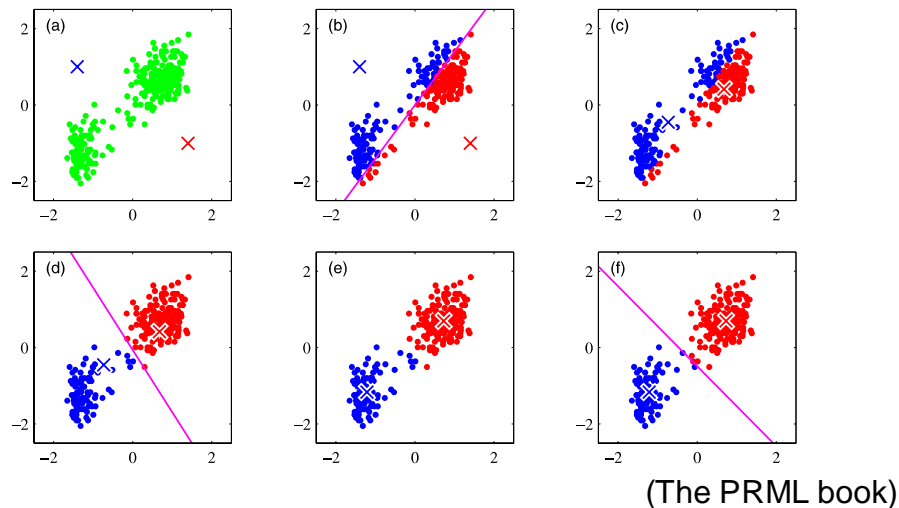
- Grouping the data points into K clusters
 - Each point has a membership to one of the clusters
 - Each cluster has a cluster center (not necessarily one of the data points)
 - The membership is determined by choosing the nearest cluster center
 - The cluster center is the mean of the data points that belong to the cluster



Learning Algorithm

- Iterative learning

- Initialize the cluster centers with random values (a)
- Compute the memberships of each data point given the cluster centers (b)
- Update the cluster centers by averaging the data points that belong to them (c)
- Repeat the two steps above until convergence (d, e, f)

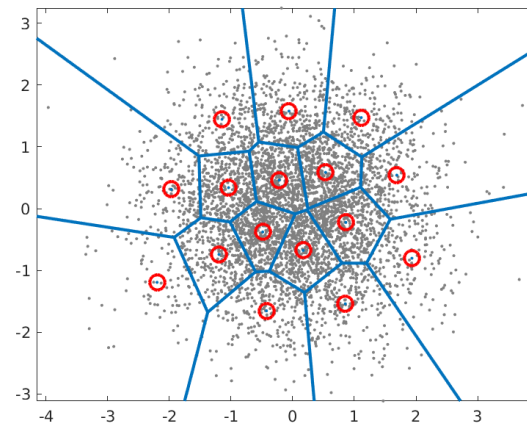


The loss monotonically decreases every iteration

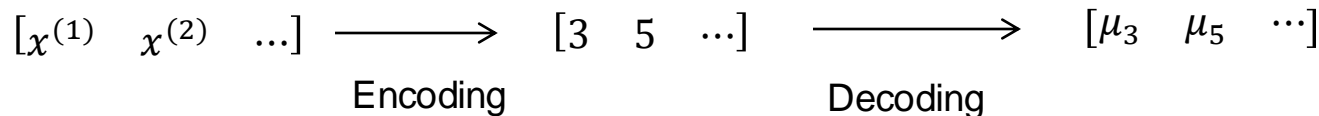
Data Compression Using K-means

- Vector Quantization

- The set of cluster centers is called “codebook”:
- Encoding a sample vector to a single scalar value of “codebook index” (membership index)
- The compressed data can be reconstructed using the codebook
- Example: CELP (Code-Excited Linear Prediction)
 - A component of speech sound is vector-quantized and the codebook index is transmitted in the speech communication

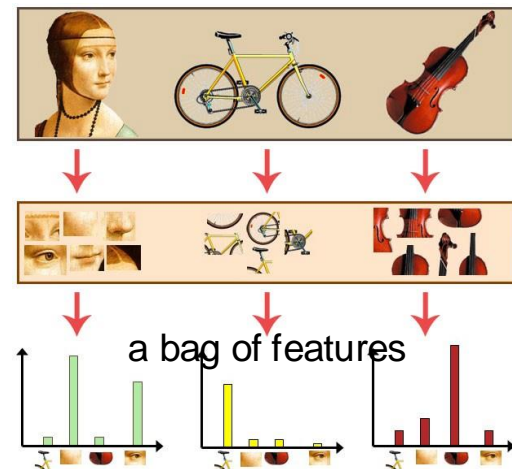
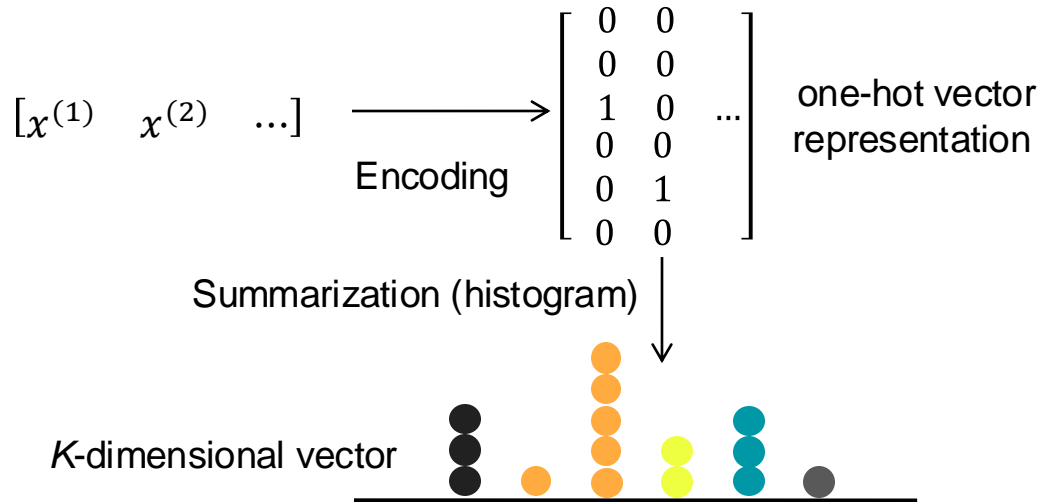


Example of a codebook for a 2D Gaussian with 16 code vectors



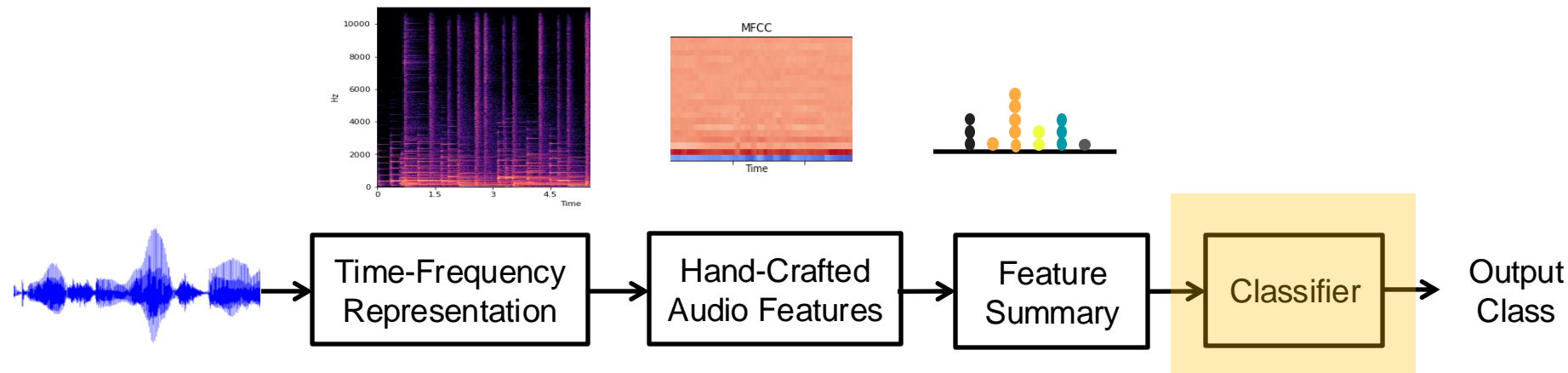
Codebook-based Feature Summarization

- Compute the histogram of codebook index
 - Represent the codebook index with one-hot vector
 - if K is a large number, it is regarded as a sparse representation of the features
 - Useful for summarizing a long sequence of feature-level features
 - Often called “a bag of features” (computer vision) or “a bag of words” (NLP)



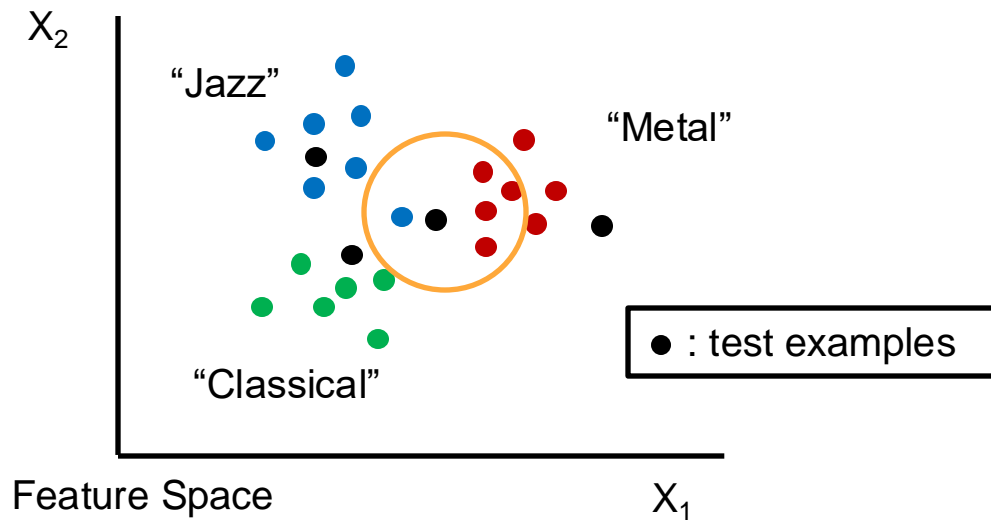
Classifiers

- Conduct supervised learning with the feature vectors and labels
 - Off-the-shelf classifiers are usually used
 - Examples: K-NN, Logistic regression, support vector machine, multi-layer-perceptron



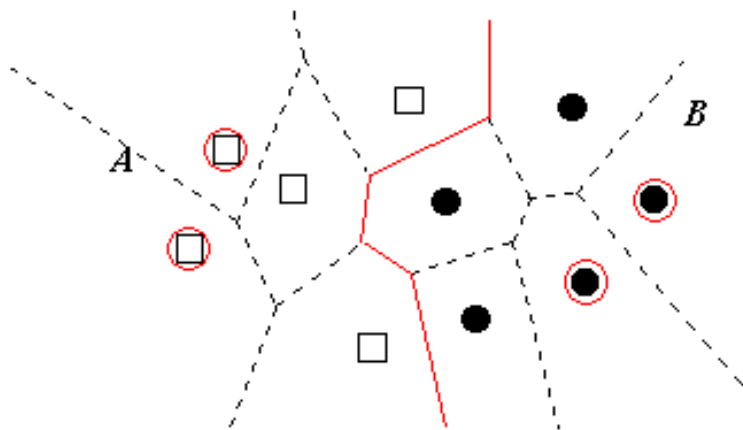
K-Nearest Neighbor (K-NN)

- Find “K” nearest training examples and do the majority voting
 - K is typically an odd number

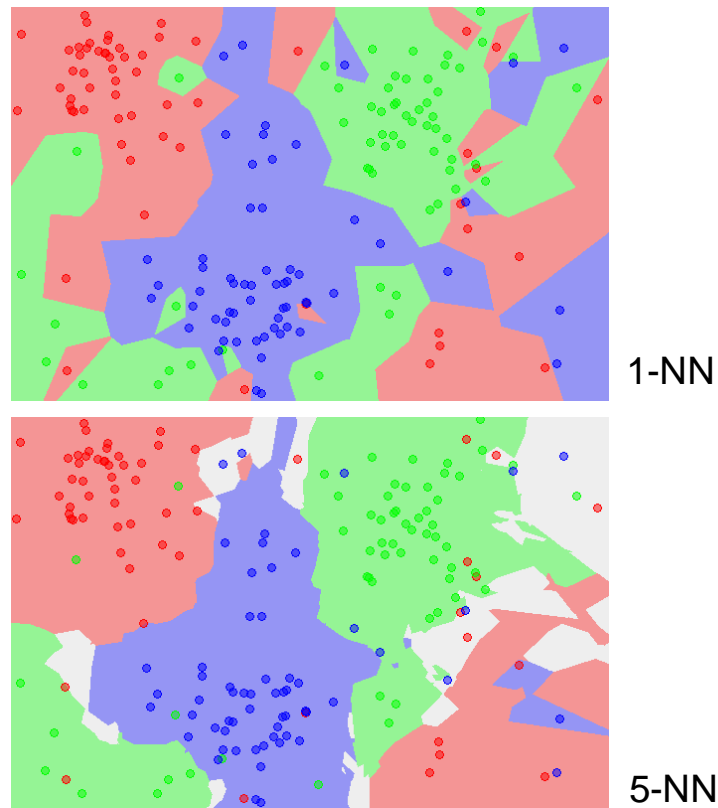


K-Nearest Neighbor (K-NN)

- Decision boundaries
 - Result in the Voronoi diagram



<http://cgm.cs.mcgill.ca/~godfried/teaching/projects.pr.98/sergei/project.html>



https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

K-Nearest Neighbor (K-NN)

- K-NN is a simple and reliable classifier
 - No model parameters: K and the distance are only options
 - Find non-linear boundaries: Voronoi diagram
- However, the distance from the test example should be computed across the entire training examples
 - Slow computation: “test phase”: $O(1)$ in train and $O(N)$ in test
 - Require a large memory to store the training data
 - Fast K-NN (usually implemented with a hash table) is another research topic to speed up computing the distance

Model-based Approach for Classification

- Learning model f is defined in terms of parameters θ and predicts the class from $f(x|\theta)$ given the input x
 - The model does not memorize the training data
- The model parameters θ are **learned** by minimizing the average loss for the training data, which is given as input data x and output label y (supervised learning)
 - There are many choices of loss functions that measure the difference between the prediction and the ground truth y

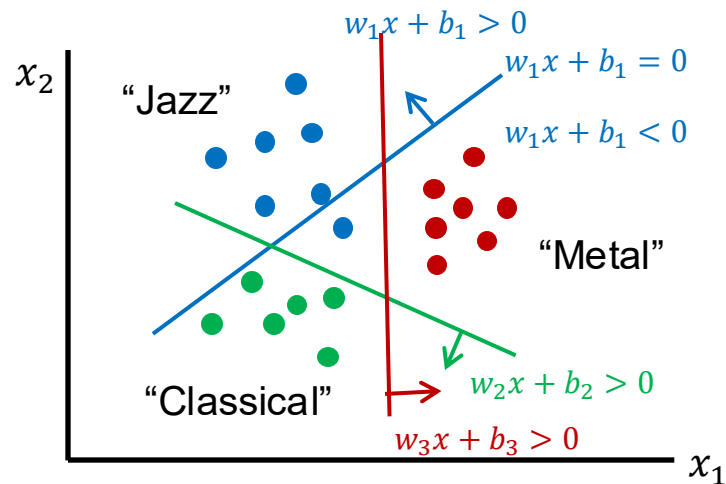
$$\min_{\theta} \frac{1}{N} \sum_{i=1}^N l(f(x_i|\theta), y_i)$$

Linear Model

- Defined as $y' = f(x|\theta) = Wx + b$

$$\begin{matrix} w_1 \\ w_2 \\ w_3 \end{matrix} \begin{bmatrix} \text{blue bar} \\ \text{green bar} \\ \text{red bar} \end{bmatrix} \begin{bmatrix} x \\ x \\ x \end{bmatrix} + \begin{bmatrix} \text{blue box} \\ \text{green box} \\ \text{red box} \end{bmatrix} \begin{matrix} b_1 \\ b_2 \\ b_3 \end{matrix} = \begin{bmatrix} \\ \\ \end{bmatrix}$$

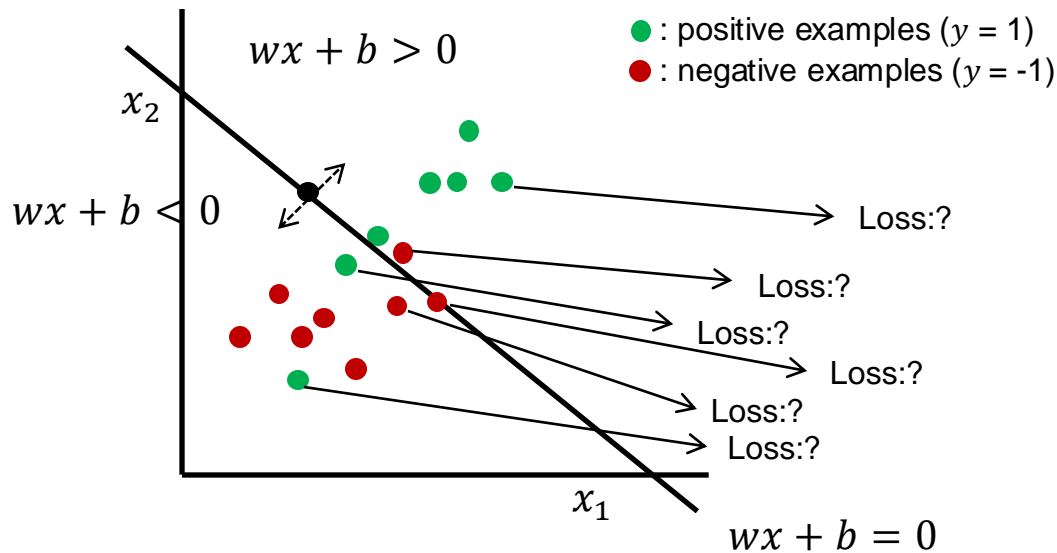
W x b $f(x; W, b)$



- Each row of W and b (w_i, b_i) is a “**class template**” or “**class prototype**”
 - It separates the examples of the corresponding class from others
- In test phase, $f(x; W, b)$ works as a score function
 - Predict the class by choosing the class template that has the maximum score from a new input: $\hat{y} = \underset{c}{\operatorname{argmax}}(w_c x + b_c)$

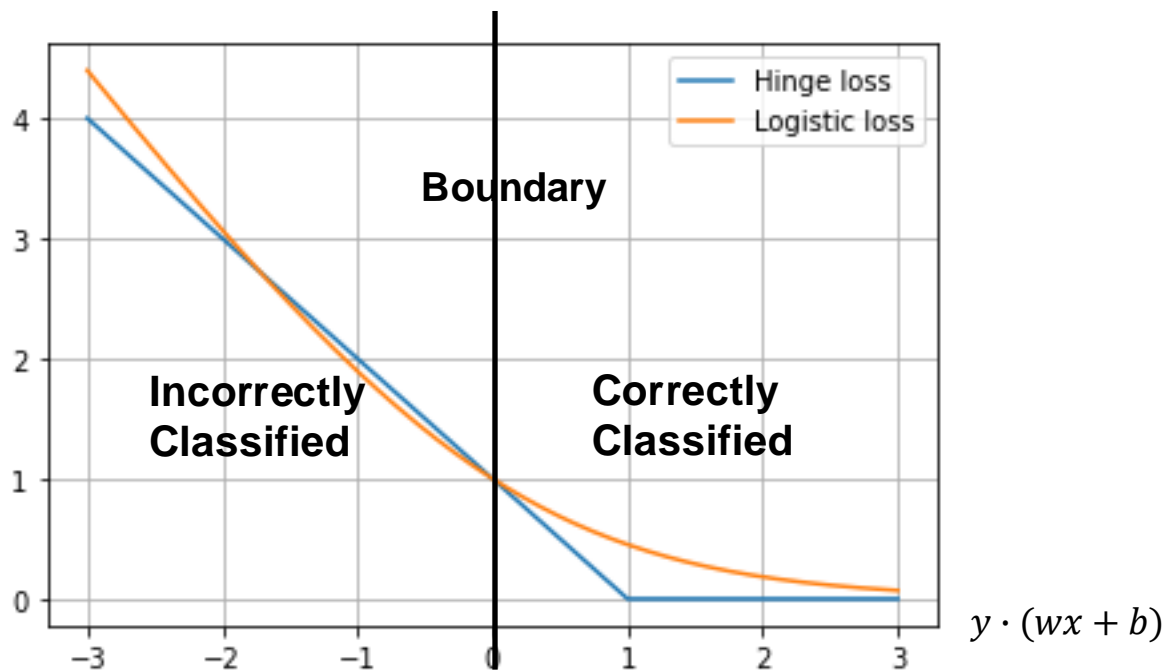
Loss functions

- Determine how much penalty is assigned to each example given W and b
 - Whether the example is correctly classified or not
 - How far the example is from the boundary
 - The penalty should be continuous at the boundary



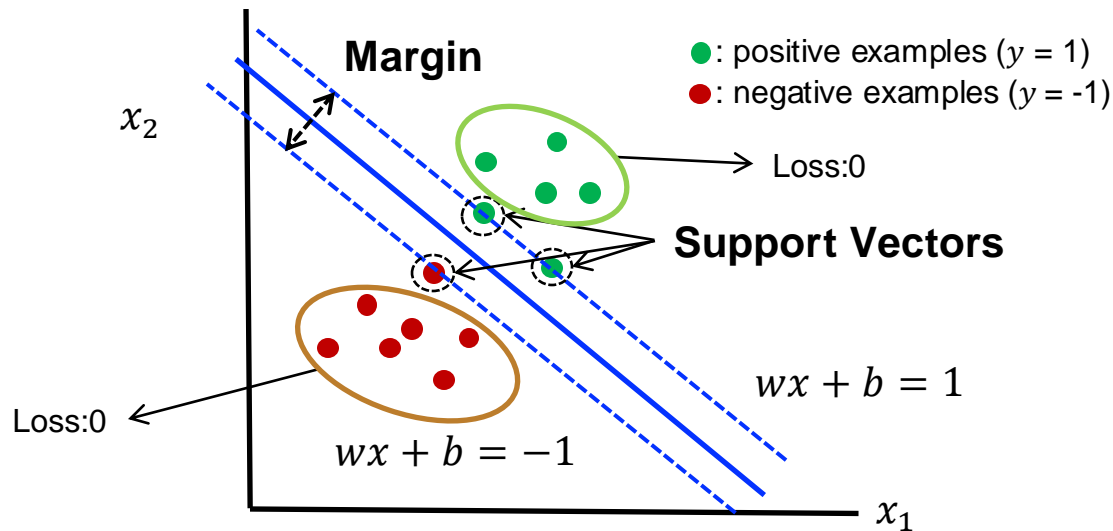
Loss functions

- Hinge loss: $l(x, y|\theta) = \max(0, 1 - y \cdot (wx + b))$
- Logistic loss: $l(x, y|\theta) = \log(1 + e^{-y \cdot (wx + b)})$



Support Vector Machine

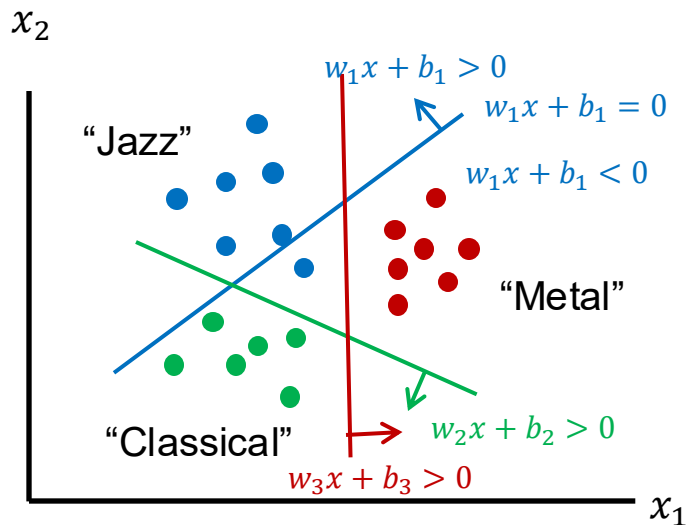
- Use the hinge loss function
 - The examples with non-zero loss are called “support vectors”. The number of support vectors is much smaller than the size of the training set
 - The boundary is determined to have the maximum margin between the support vectors (“maximum margin classifier”)



Multi-Class SVM

- The hinge loss function is generalized to multi-class classification

$$f(x, y = k | \theta) = \sum_{c \neq k} \max(0, 1 - w_k x - b_k + \underbrace{w_c x + b_c}_{\text{Loss contributions from the others}})$$



Logistic Regression

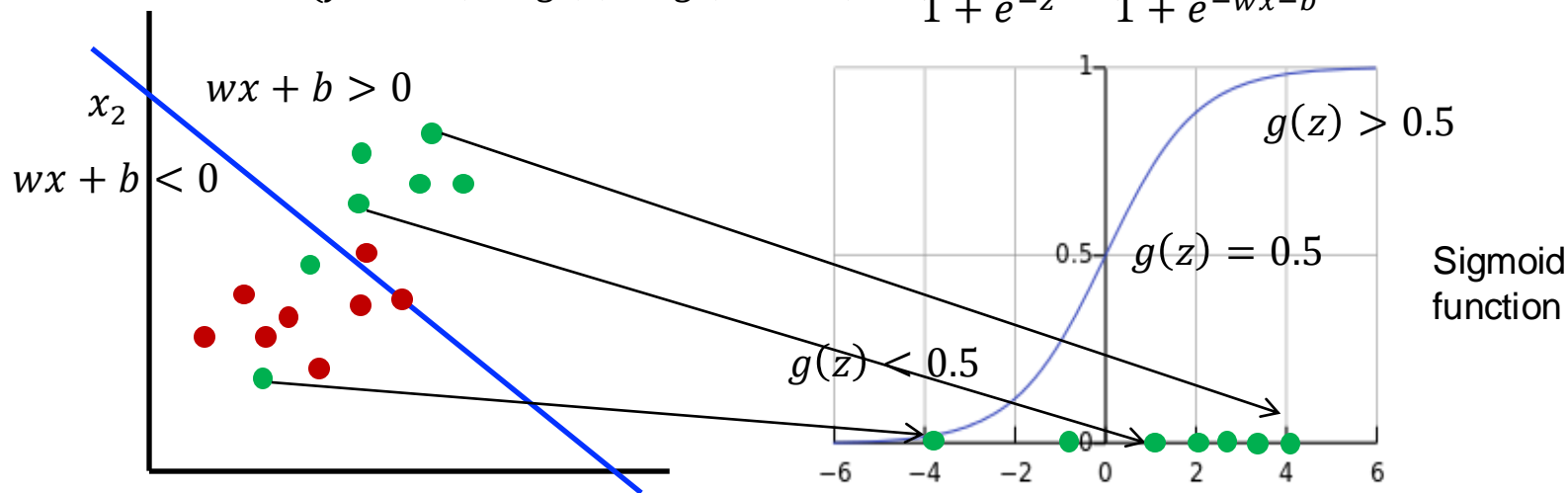
- Defining the prediction from the conditional probability $P(y|x)$.

- For the binary classification, we use Bernoulli distribution

$$P(y = 1|x) = g, P(y = -1|x) = 1 - g$$

- Parameterize g with the linear function of the input and sigmoid function

$$P(y = 1|x) = g(z) = g(wx + b) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-wx - b}}$$



Logistic Regression

- The logistic loss is defined as the cross-entropy between the probability distribution and the ground truth (or true distribution)
 - This also corresponds to the negative log-likelihood
 - Minimizing the negative log-likelihood → **“Maximum likelihood estimation”**

$$l(x, y|\theta) = \sum_c q(x) [-\log p(x)] = - \sum_c q(x) \cdot \log g(w_c x + b) \quad q(x)=[1,0] \text{ or } [0,1]$$

$$= \underbrace{-\log g(wx + b)}_{\text{positive examples}} - \underbrace{\log(1 - g(wx + b))}_{\text{negative examples}} = \log(1 + e^{-y \cdot (wx + b)})$$

positive examples negative examples

Multi-Class Logistic Regression

- Softmax: multi-class extension of the logistic regression

- From the Bernoulli distribution to the multinomial distribution

$$P(y = k|x) = \frac{e^{w_k x + b_k}}{\sum_c e^{w_c x + b_c}}$$

“Logit”

$$w_1 x + b_1$$

$$w_2 x + b_2$$

$$w_3 x + b_3$$

Exponential

$$e^{w_1 x + b_1}$$

$$e^{w_2 x + b_2}$$

$$e^{w_3 x + b_3}$$

Normalization

$$\frac{e^{w_1 x + b_1}}{\sum_c e^{w_c x + b_c}}$$

$$\frac{e^{w_2 x + b_2}}{\sum_c e^{w_c x + b_c}}$$

$$\frac{e^{w_3 x + b_3}}{\sum_c e^{w_c x + b_c}}$$

Softmax of “Logit”

- Again, the loss function is defined as the cross-entropy between the multinomial distribution and the ground truth

- Also, it is the same as the negative log-likelihood of the training data
- Minimizing the negative log-likelihood → **“Maximum likelihood estimation”**

$$l(x, y = k|\theta) = -(w_k x + b_k) + \log\left(\sum_c e^{w_c x + b_c}\right)$$

Maximum Likelihood Estimation (MLE)

- For independent and identically distributed (IID) random variables,
 - MLE finds the values of model parameters that maximizes the likelihood function $L(\theta; x, y) = p(y|x, \theta) = \prod_i p((y_i|x_i, \theta))$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} L(\theta; x, y) = \underset{\theta}{\operatorname{argmax}} \prod_i p((y_i|x_i, \theta))$$

Converting to the log scale
due to the numerical reason

$$= \underset{\theta}{\operatorname{argmin}} -\log(\prod_i p((y_i|x_i, \theta))) = \underset{\theta}{\operatorname{argmin}} \sum_i -\log(p((y_i|x_i, \theta)))$$

		$x_1, y_1 = \text{jazz}$	$x_2, y_2 = \text{classical}$	$x_3, y_3 = \text{metal}$	$x_4, y_4 = \text{jazz}$
θ_1	$p(\text{jazz} x)$	0.8	0.2	0.2	0.3
	$p(\text{classical} x)$	0.1	0.7	0.3	0.4
	$p(\text{metal} x)$	0.1	0.1	0.5	0.3
θ_2	$p(\text{jazz} x)$	0.7	0.2	0.3	0.2
	$p(\text{classical} x)$	0.2	0.6	0.2	0.3
	$p(\text{metal} x)$	0.1	0.2	0.5	0.5

Which θ
maximizes the
likelihood more?

Learning: Gradient descent

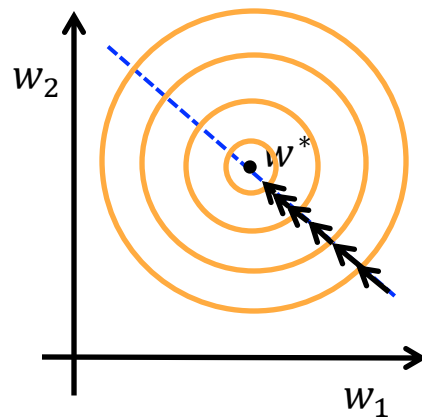
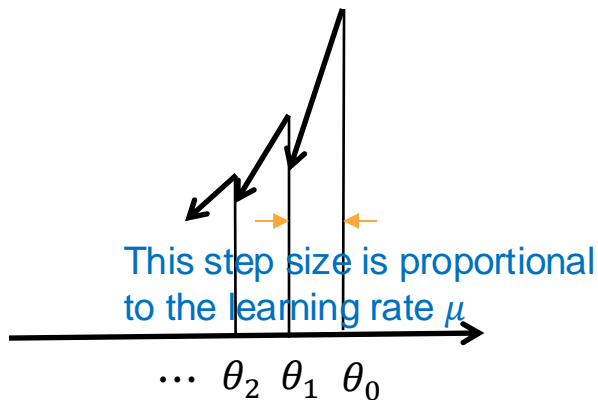
- Update rule for learning (or optimization)

$$\theta_j = \theta_{j-1} - \mu \left. \frac{\partial l(\theta)}{\partial \theta} \right|_{\theta = \theta_{j-1}}$$

$\frac{\partial l(\theta)}{\partial \theta}$: gradient of the loss function (vector)
 μ : learning rate

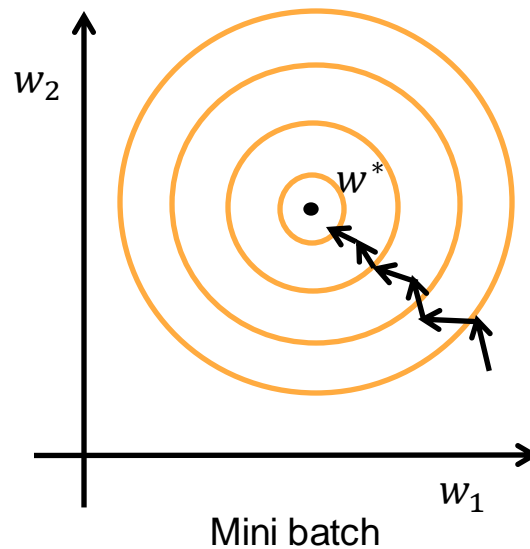
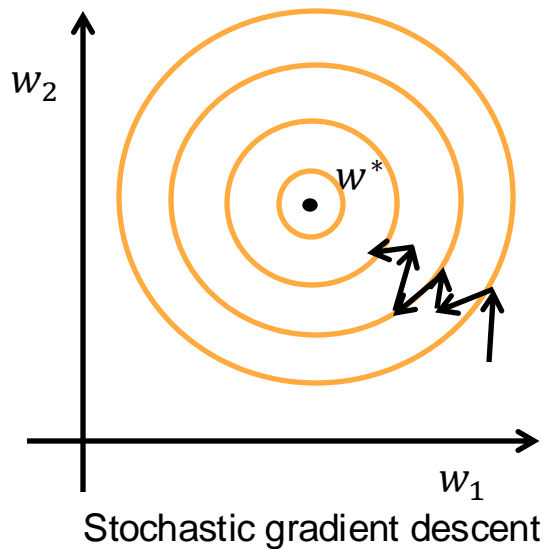
$$l(\theta) = \sum_{i=1}^N f(x_j, y_i | \theta)$$

- θ is usually initialized with random values
- As the iteration goes on, the parameters move towards local minima



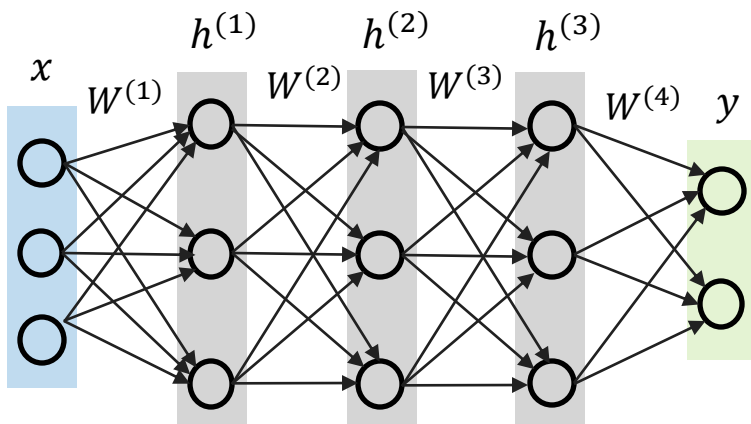
Learning Parameters

- Training Data Size
 - Stochastic gradient descent (SGD): $N = 1$
 - Mini batch: $N =$ a small number of sampled data (e.g. 32, 64, 128 samples)
 - Batch update: use the entire training data



Multi-Layer Perceptron (MLP)

- A stack of linear transform and a non-linear function
 - $g_i(x)$ is a non-linear function such as sigmoid, tanh, ReLU
 - This transforms the linear boundary into a non-linear boundary
 - This is the basic form of neural networks



$l(y, \hat{y})$

$$y = W^{(4)}h^{(3)} + b^{(4)}$$

$$h^{(3)} = g(z^{(3)})$$

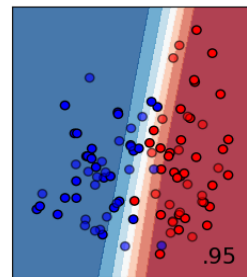
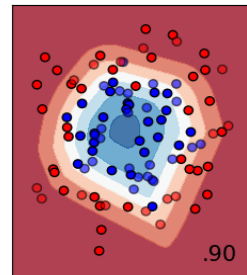
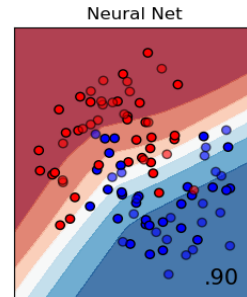
$$z^{(3)} = W^{(3)}h^{(2)} + b^{(3)}$$

$$h^{(2)} = g(z^{(2)})$$

$$z^{(2)} = W^{(2)}h^{(1)} + b^{(2)}$$

$$h^{(1)} = g(z^{(1)})$$

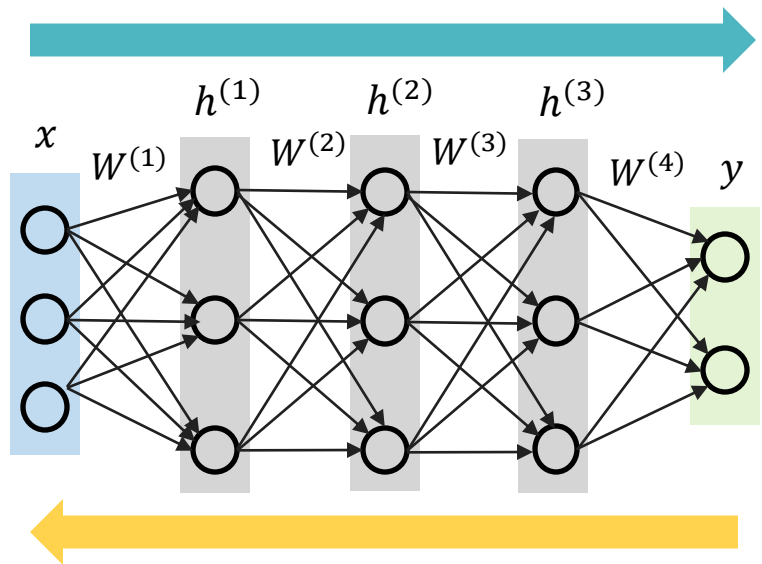
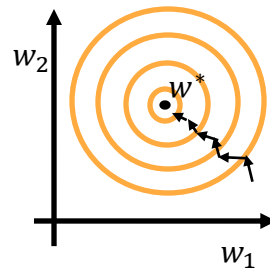
$$z^{(1)} = W^{(1)}x + b^{(1)}$$



Training Neural Networks

- Repeating the feedforward and backward passes
 - Updating the weights with the gradients

$$W^{(l)new} = W^{(l)old} - \mu \frac{\partial l(W^{(l)old})}{\partial W^{(l)old}}$$



$l(y, \hat{y})$

$$y = W^{(4)}h^{(3)} + b^{(4)}$$

$$h^{(3)} = g(z^{(3)})$$

$$z^{(3)} = W^{(3)}h^{(2)} + b^{(3)}$$

$$h^{(2)} = g(z^{(2)})$$

$$z^{(2)} = W^{(2)}h^{(1)} + b^{(2)}$$

$$h^{(1)} = g(z^{(1)})$$

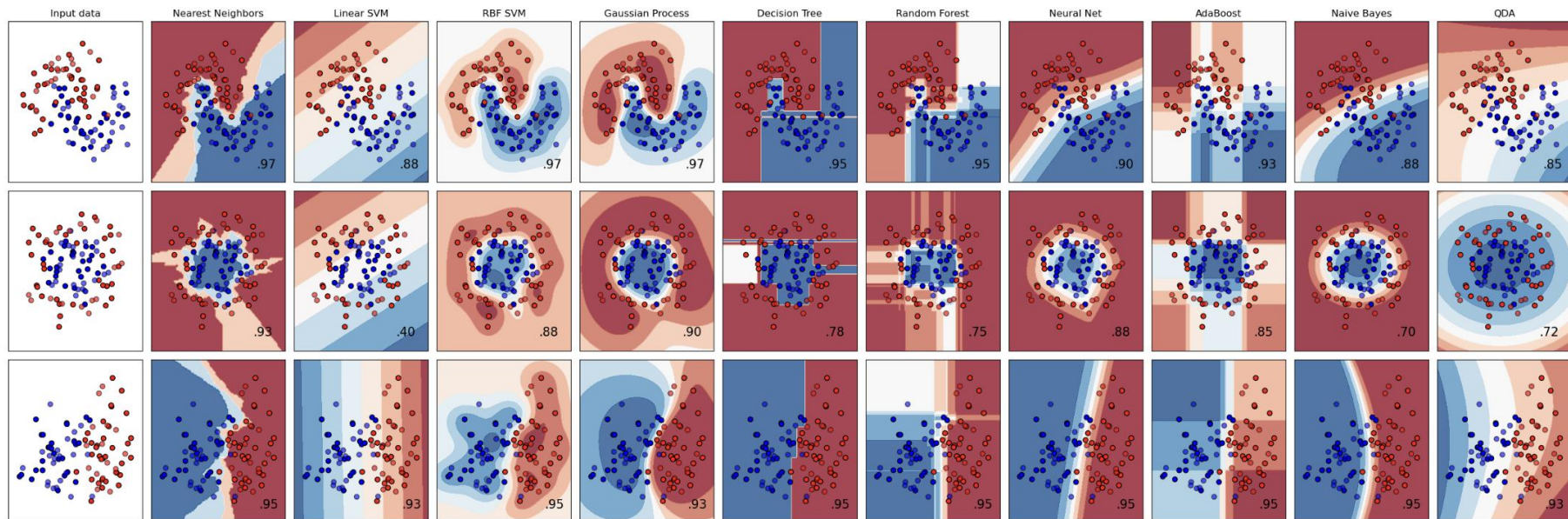
$$z^{(1)} = W^{(1)}x + b^{(1)}$$

Visualizing Supervised Learning

- <https://playground.tensorflow.org/>

Other Commonly Used Traditional Classifiers

- SVM with kernels, Random Forest, Gaussian Mixture Model, ...



Musical Genre Classification of Audio Signals

George Tzanetakis, *Student Member, IEEE*, and Perry Cook, *Member, IEEE*

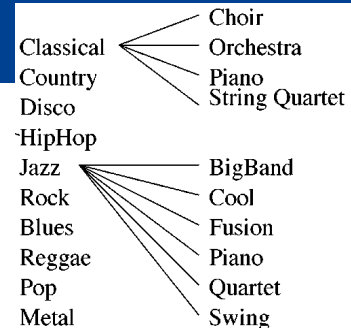
- The GTZAN paper
 - The most representative paper in music genre classification
 - Used various audio features + KNN/GMM classifiers
 - Most cited paper in MIR research (about 4295 as of Feb-25, 2024)
 - Released the GTZAN dataset (10 genre, 100 songs per genre)
 - The accuracy is not very high (~ 60%)
 - A number of papers improved it to 90% or so
 - Released the software (MARSYAS)
 - However, the dataset was criticized due to repetition, mislabeling and audio distortion...

Case Study: Audio Features

- Timbral texture features: summarized in 1 sec of window
 - Spectral centroid, spectral roll-off, spectral flux
 - Zero-crossing rate, Low-energy features
 - MFCC
- Pitch / Tonal features
 - Based on a multi-pitch estimation method using summary enhanced auto-correlation function (SACF).
 - Three dominant peaks in the SACF that correspond to pitch values are accumulated as a histogram (pitch histogram)
- Rhythmic features
 - Beat histogram

Case Study: Results

- 10-fold cross validation
 - 90% for training and 10% for testing



CLASSIFICATION ACCURACY MEAN AND STANDARD DEVIATION

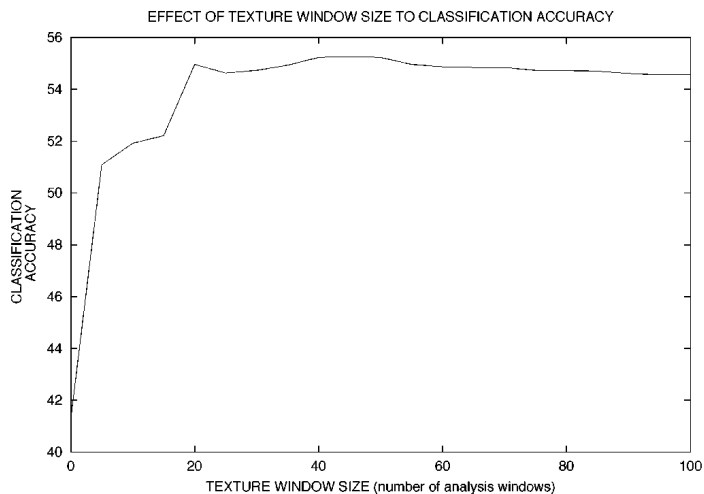
	Genres(10)	Classical(4)	Jazz(6)
Random	10	25	16
RT GS	44 ± 2	61 ± 3	53 ± 4
GS	59 ± 4	77 ± 6	61 ± 8
GMM(2)	60 ± 4	81 ± 5	66 ± 7
GMM(3)	61 ± 4	88 ± 4	68 ± 7
GMM(4)	61 ± 4	88 ± 5	62 ± 6
GMM(5)	61 ± 4	88 ± 5	59 ± 6
KNN(1)	59 ± 4	77 ± 7	57 ± 6
KNN(3)	60 ± 4	78 ± 6	58 ± 7
KNN(5)	56 ± 3	70 ± 6	56 ± 6

GENRE CONFUSION MATRIX

	cl	co	di	hi	ja	ro	bl	re	po	me
cl	69	0	0	0	1	0	0	0	0	0
co	0	53	2	0	5	8	6	4	2	0
di	0	8	52	11	0	13	14	5	9	6
hi	0	3	18	64	1	6	3	26	7	6
ja	26	4	0	0	75	8	7	1	2	1
ro	5	13	4	1	9	40	14	1	7	33
bl	0	7	0	1	3	4	43	1	0	0
re	0	9	10	18	2	12	11	59	7	1
po	0	2	14	5	3	5	0	3	66	0
me	0	1	0	1	0	4	2	0	0	53

Case Study: Results

- Importance of texture window size and individual features



(1 second \approx 40 frames)

INDIVIDUAL FEATURE SET IMPORTANCE

	Genres	Classical	Jazz
RND	10	25	16
PHF(5)	23	40	26
BHF(6)	28	39	31
STFT(9)	45	78	58
MFCC(10)	47	61	56
FULL(30)	59	77	61