Machine Learning 3/3

Lecture 09

Computer Vision for Geosciences

2021-05-07

PCA toy example

*We have several wine bottles in our cellar, 11 **features** (alcohol, acidity, etc.) describe its **quality**.*
Which features best define it, are there related features (i.e. covariant) which are redundent?

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

PCA toy example

*We have several wine bottles in our cellar, 11 **features** (alcohol, acidity, etc.) describe its **quality**. Which features best define it, are there related features (i.e. covariant) which are redundent?*

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

$\Rightarrow$ PCA allows to summarize each wine with fewer characteristics
$\Rightarrow$ **reduce data dimensions**

PCA toy example

*We have several wine bottles in our cellar, 11 **features** (alcohol, acidity, etc.) describe its **quality**. Which features best define it, are there related features (i.e. covariant) which are redundent?*

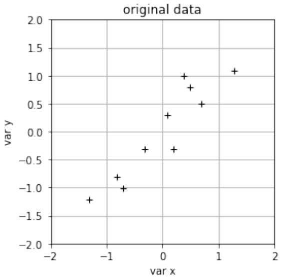
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

$\Rightarrow$ PCA allows to summarize each wine with fewer characteristics
$\Rightarrow$ **reduce data dimensions**

$\Rightarrow$ PCA does *not* select some features and discards others,
instead it **defines new features** (using linear combinations of available features)
which will best represent wine variability

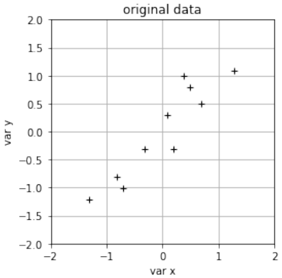
$\Rightarrow$ Example with 2 variables: *compute covariance matrix $\rightarrow$ find largest eigenvalues & eigenvectors $\rightarrow$ project*

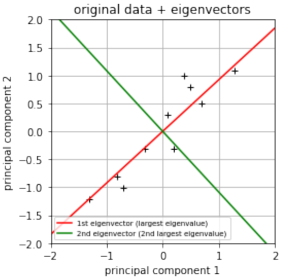
(**eigenvectors** represent the directions of the largest variance of the data, **eigenvalues** represent the magnitude of this variance in those directions)



⇒ Example with 2 variables: *compute covariance matrix → find largest eigenvalues & eigenvectors → project*

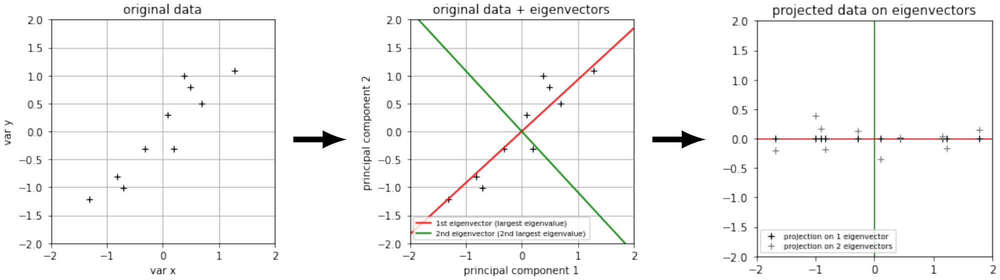
(**eigenvectors** represent the directions of the largest variance of the data, **eigenvalues** represent the magnitude of this variance in those directions)





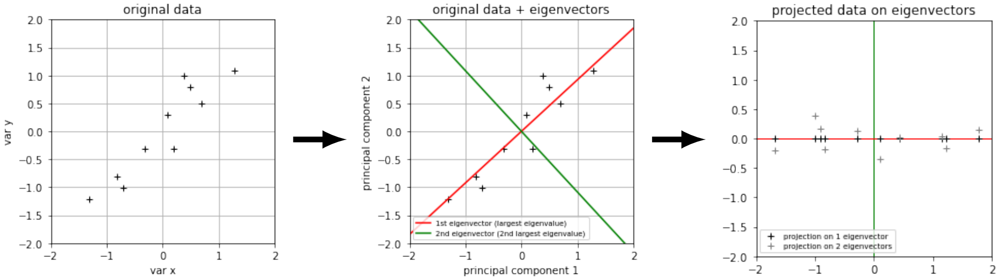
⇒ Example with 2 variables: *compute covariance matrix → find largest eigenvalues & eigenvectors → project*

(**eigenvectors** represent the directions of the largest variance of the data, **eigenvalues** represent the magnitude of this variance in those directions)



⇒ Example with 2 variables: *compute covariance matrix → find largest eigenvalues & eigenvectors → project*

(**eigenvectors** represent the directions of the largest variance of the data, **eigenvalues** represent the magnitude of this variance in those directions)



→ low variance (dispersion) of the data along the 2nd eigenvector

⇒ the 2 original features (var x, var y) could be reduced to 1 feature, i.e. the projection on the 1st eigenvector

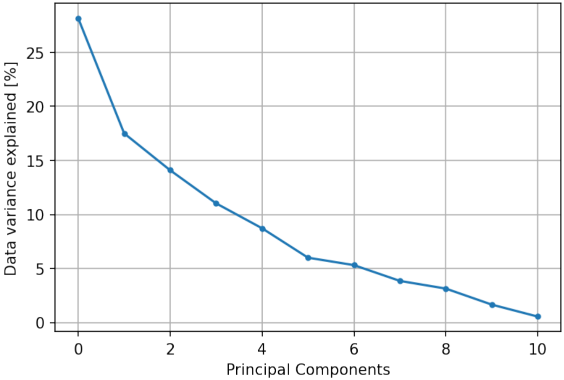
$\Rightarrow$ Do the same with the 11 features

$\rightarrow$ search for the principal components in a 11-dimensional space *(the max. number of components is restricted by the number of features)*

⇒ Do the same with the 11 features

→ search for the principal components in a 11-dimensional space *(the max. number of components is restricted by the number of features)*

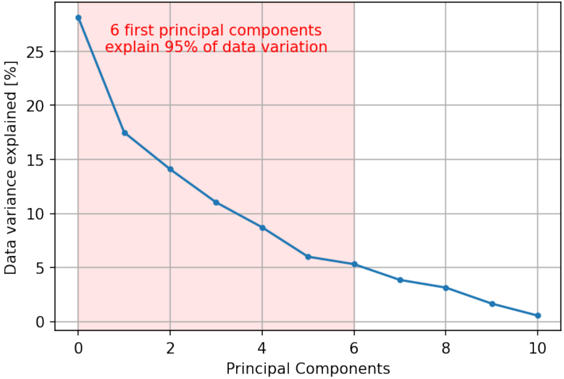
Q1: How much data variance is explained by each principal component (eigenvector)?



⇒ Do the same with the 11 features

→ search for the principal components in a 11-dimensional space *(the max. number of components is restricted by the number of features)*

Q1: How much data variance is explained by each principal component (eigenvector)?



⇒ Do the same with the 11 features

→ search for the principal components in a 11-dimensional space *(the max. number of components is restricted by the number of features)*

Q1: How much data variance is explained by each principal component (eigenvector)?
Q2: How do the 11 eigenvectors (PCs) relate to the original feature space?

	0	1	2	3	4	5	6	7	8	9	10
0	0.489314	-0.238584	0.463632	0.146107	0.212247	-0.036158	0.023575	0.395353	-0.438520	0.242921	-0.113232
1	-0.110503	0.274930	-0.151791	0.272080	0.148052	0.513567	0.569487	0.233575	0.006711	-0.037554	-0.386181
2	-0.123302	-0.449963	0.238247	0.101283	-0.092614	0.428793	0.322415	-0.338871	0.057697	0.279786	0.471673
3	-0.229617	0.078960	-0.079418	-0.372793	0.666195	-0.043538	-0.034577	-0.174500	-0.003788	0.550872	-0.122181
4	-0.082614	0.218735	-0.058573	0.732144	0.246501	-0.159152	-0.222465	0.157077	0.267530	0.225962	0.350681
5	0.101479	0.411449	0.069593	0.049156	0.304339	-0.014000	0.136308	-0.391152	-0.522116	-0.381263	0.361645
6	-0.350227	-0.533735	0.105497	0.290663	0.370413	-0.116596	-0.093662	-0.170481	-0.025138	-0.447469	-0.327651
7	-0.177595	-0.078775	-0.377516	0.299845	-0.357009	-0.204781	0.019036	-0.239223	-0.561391	0.374604	-0.217626
8	-0.194021	0.129110	0.381450	-0.007523	-0.111339	-0.635405	0.592116	-0.020719	0.167746	0.058367	-0.037603
9	-0.249523	0.365925	0.621677	0.092872	-0.217671	0.248483	-0.370750	-0.239990	-0.010970	0.112320	-0.303015
10	0.639691	0.002389	-0.070910	0.184030	0.053065	-0.051421	0.068702	-0.567332	0.340711	0.069555	-0.314526

⇒ Do the same with the 11 features

→ search for the principal components in a 11-dimensional space *(the max. number of components is restricted by the number of features)*

Q1: How much data variance is explained by each principal component (eigenvector)?
Q2: How do the 11 eigenvectors (PCs) relate to the original feature space?

	0	1	2	3	4	5	6	7	8	9	10
0	0.489314	-0.238584	0.463632	0.146107	0.212247	-0.036158	0.023575	0.395353	-0.438520	0.242921	-0.113232
1	-0.110503	0.274930	-0.151791	0.272080	0.148052	0.513567	0.569487	0.233575	0.006711	-0.037554	-0.386181
2	-0.123302	-0.449963	0.238247	0.101283	-0.092614	0.428793	0.322415	-0.338871	0.057697	0.279786	0.471673
3	-0.229617	0.078960	-0.079418	-0.372793	0.666195	-0.043538	-0.034577	-0.174500	-0.003788	0.550872	-0.122181
4	-0.082614	0.218735	-0.058573	0.732144	0.246501	-0.159152	-0.222465	0.157077	0.267530	0.225962	0.350681
5	0.101479	0.411449	0.069593	0.049156	0.304339	-0.014000	0.136308	-0.391152	-0.522116	-0.381263	0.361645
6	-0.350227	-0.533735	0.105497	0.290663	0.370413	-0.116596	-0.093662	-0.170481	-0.025138	-0.447469	-0.327651
7	-0.177595	-0.078775	-0.377516	0.299845	-0.357009	-0.204781	0.019036	-0.239223	-0.561391	0.374604	-0.217626
8	-0.194021	0.129110	0.381450	-0.007523	-0.111339	-0.635405	0.592116	-0.020719	0.167746	0.058367	-0.037603
9	-0.249523	0.365925	0.621677	0.092872	-0.217671	0.248483	-0.370750	-0.239990	-0.010970	0.112320	-0.303015
10	0.639691	0.002389	-0.070910	0.184030	0.053065	-0.051421	0.068702	-0.567332	0.340711	0.069555	-0.314526

Principal Component 1

PC 1 *= 0.49*feature0 + -0.24*feature1 + 0.46*feature2 + 0.15*feature3 + 0.21*feature4 + -0.04*feature5 + 0.02*feature6 + 0.40*feature7 + -0.44*feature8 + 0.24*feature9 + -0.11*feature10*

⇒ Do the same with the 11 features

→ search for the principal components in a 11-dimensional space *(the max. number of components is restricted by the number of features)*

Q1: How much data variance is explained by each principal component (eigenvector)?
Q2: How do the 11 eigenvectors (PCs) relate to the original feature space?
Q3: How accurate is the prediction using *all original 11 features*, versus using *only the e.g. 6 first principal components*?

$\Rightarrow$ Do the same with the 11 features

$\rightarrow$ search for the principal components in a 11-dimensional space *(the max. number of components is restricted by the number of features)*

Q1: How much data variance is explained by each principal component (eigenvector)?
Q2: How do the 11 eigenvectors (PCs) relate to the original feature space?
Q3: How accurate is the prediction using *all original 11 features*, versus using *only the e.g. 6 first principal components*?

Prediction accuracy of wine *quality* (classification task using KNN):

- using 11 original features $\Rightarrow$ accuracy $= 0.79$
- using 6 first principal components $\Rightarrow$ accuracy $= 0.78$

⇒ Do the same with the 11 features

→ search for the principal components in a 11-dimensional space *(the max. number of components is restricted by the number of features)*

Q1: How much data variance is explained by each principal component (eigenvector)?
Q2: How do the 11 eigenvectors (PCs) relate to the original feature space?
Q3: How accurate is the prediction using *all original 11 features*, versus using *only the e.g. 6 first principal components*?

Prediction accuracy of wine *quality* (classification task using KNN):

- using 11 original features ⇒ accuracy = 0.79
- using 6 first principal components ⇒ accuracy = 0.78

⇒ PCA can successfully reduce data dimensionality,
and achieve (almost) the same prediction accuracy with fewer features

$\Rightarrow$ Do the same with the 11 features

$\rightarrow$ search for the principal components in a 11-dimensional space *(the max. number of components is restricted by the number of features)*

Q1: How much data variance is explained by each principal component (eigenvector)?
Q2: How do the 11 eigenvectors (PCs) relate to the original feature space?
Q3: How accurate is the prediction using *all original 11 features*, versus using *only the e.g. 6 first principal components*?

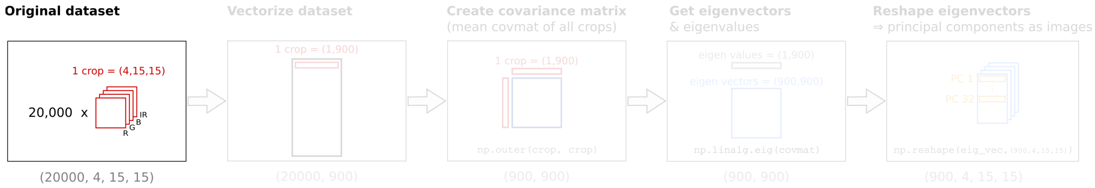
Prediction accuracy of wine *quality* (classification task using KNN):

- using 11 original features $\Rightarrow$ accuracy $= 0.79$
- using 6 first principal components $\Rightarrow$ accuracy $= 0.78$

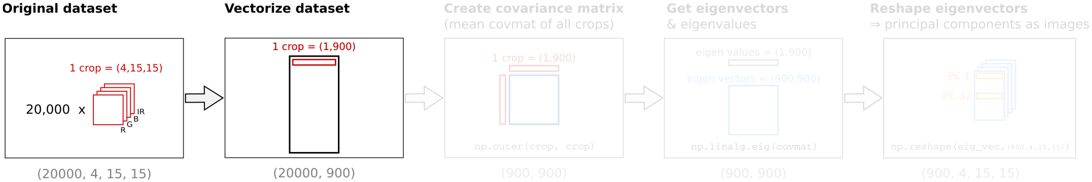
$\Rightarrow$ PCA can successfully reduce data dimensionality,
and achieve (almost) the same prediction accuracy with fewer features

$\Rightarrow$ how about using PCA on images?
$\rightarrow$ Sentinel-2 example: reduce a space with 20,000$\times$4$\times$15$\times$15 pixels (900 dimensions)

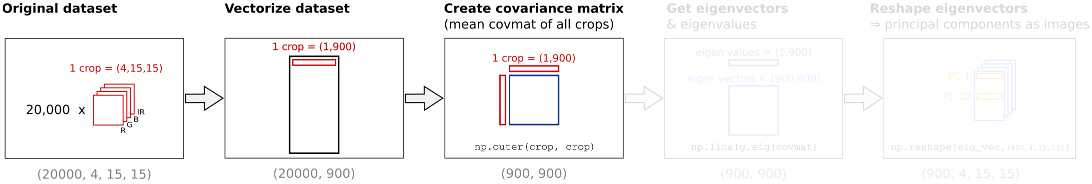
Sentinel-2 example ⇒ apply PCA on satellite image crops



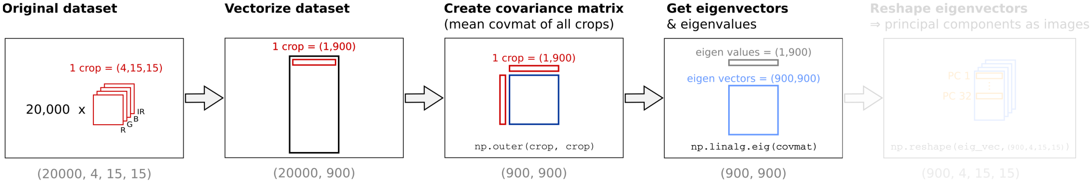
Sentinel-2 example ⇒ apply PCA on satellite image crops



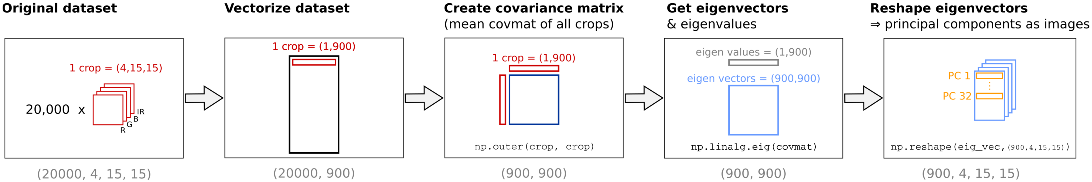
Sentinel-2 example ⇒ apply PCA on satellite image crops

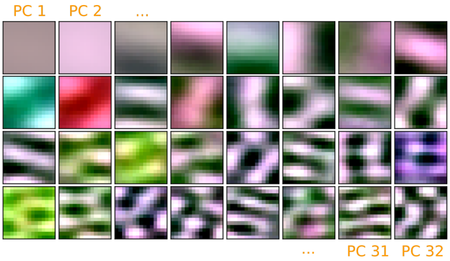


Sentinel-2 example ⇒ apply PCA on satellite image crops

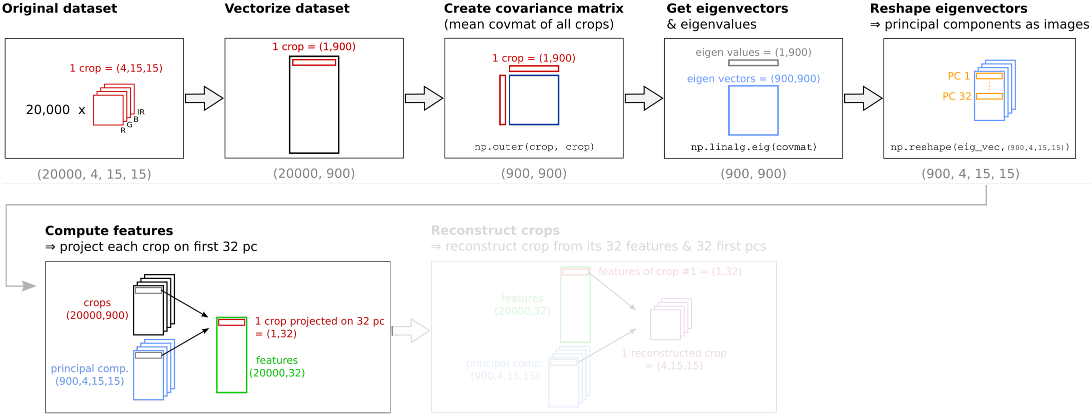


Sentinel-2 example ⇒ apply PCA on satellite image crops





Sentinel-2 example ⇒ apply PCA on satellite image crops

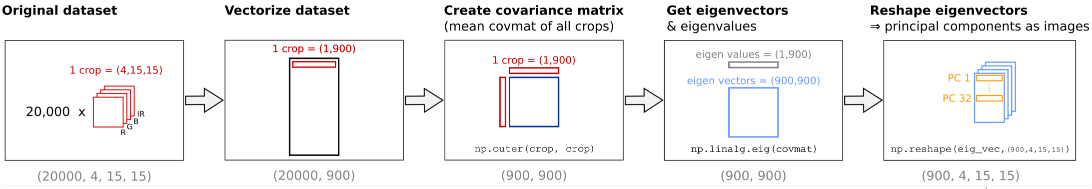


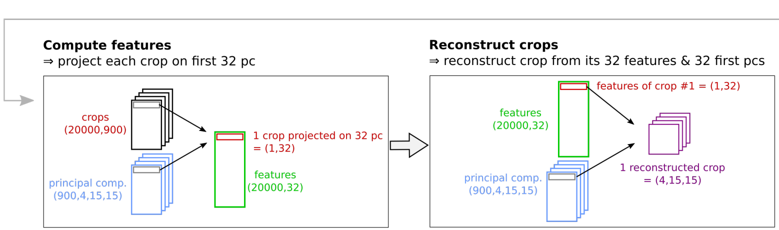

```
for i in range(20000):    #loop crops
    for j in range(32):   #loop pcs
        features[i,j] = np.dot(crop[i,:::], pc[j,:::])
                      = (900,1) @ (1,900)
                      = (scalar)
```

Reconstruction crop #1:

```
reconstruction = mean_crops   #(4, 15, 15)
for i in range(32):           #loop crop features/pcs
    reconstruction += features[0,i] * pc[i,:::]
                    = (scalar) * (4,15,15)
                    = (4,5,15)
```

Sentinel-2 example ⇒ apply PCA on satellite image crops





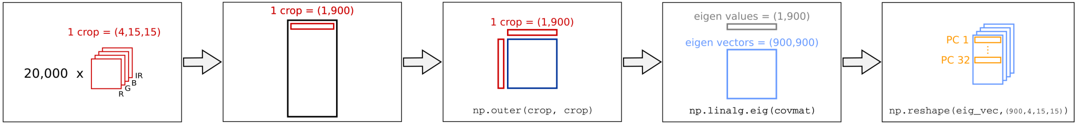

```
for i in range(20000):    #loop crops
   for j in range(32):    #loop pcs
      features[i,j] = np.dot(crop[i,:::], pc[j,:::])
                    = (900,1) @ (1,900)
                    = (scalar)
```

Reconstruction crop #1:

```
reconstruction = mean_crops    #(4, 15, 15)
for i in range(32):            #loop crop features/pcs
   reconstruction += features[0,i] * pc[i,:::]
                   = (scalar) * (4,15,15)
                   = (4,5,15)
```

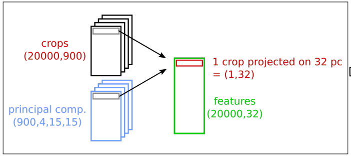
Sentinel-2 example ⇒ apply PCA on satellite image crops

Original dataset | **Vectorize dataset** | **Create covariance matrix** (mean covmat of all crops) | **Get eigenvectors** & eigenvalues | **Reshape eigenvectors** ⇒ principal components as images



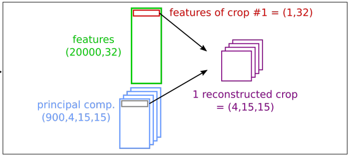
(20000, 4, 15, 15) | (20000, 900) | (900, 900) | (900, 900) | (900, 4, 15, 15)

Compute features
⇒ project each crop on first 32 pc




```
for i in range(20000):   #loop crops
   for j in range(32):   #loop pcs
      features[i,j] = np.dot(crop[i,:::], pc[j,:::])
                    = (900,1) @ (1,900)
                    = (scalar)
```

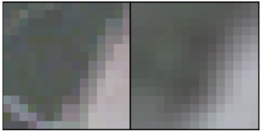
Reconstruct crops
⇒ reconstruct crop from its 32 features & 32 first pcs



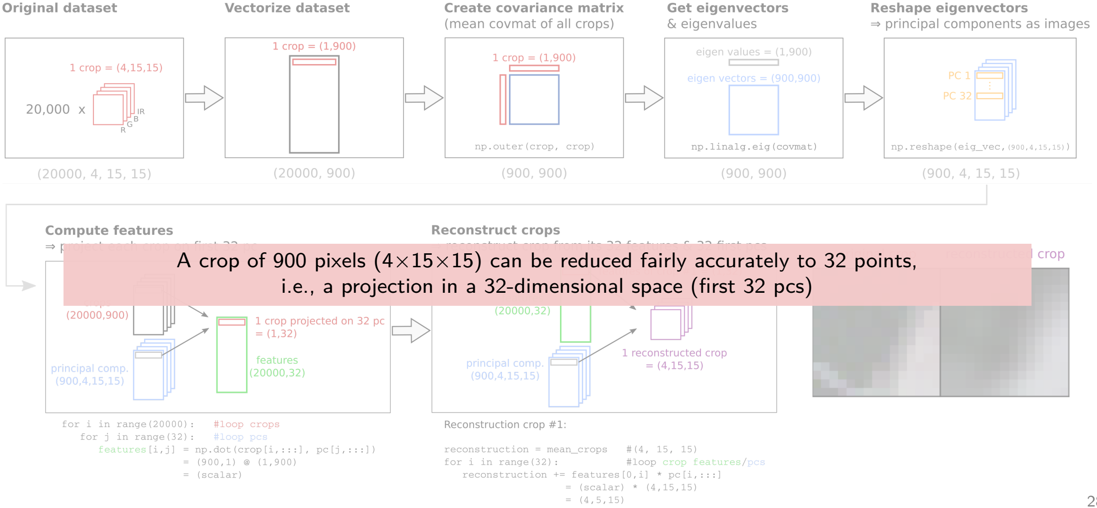
Reconstruction crop #1:

```
reconstruction = mean_crops    #(4, 15, 15)
for i in range(32):            #loop crop features/pcs
   reconstruction += features[0,i] * pc[i,:::]
                   = (scalar) * (4,15,15)
                   = (4,5,15)
```

original crop | reconstructed crop

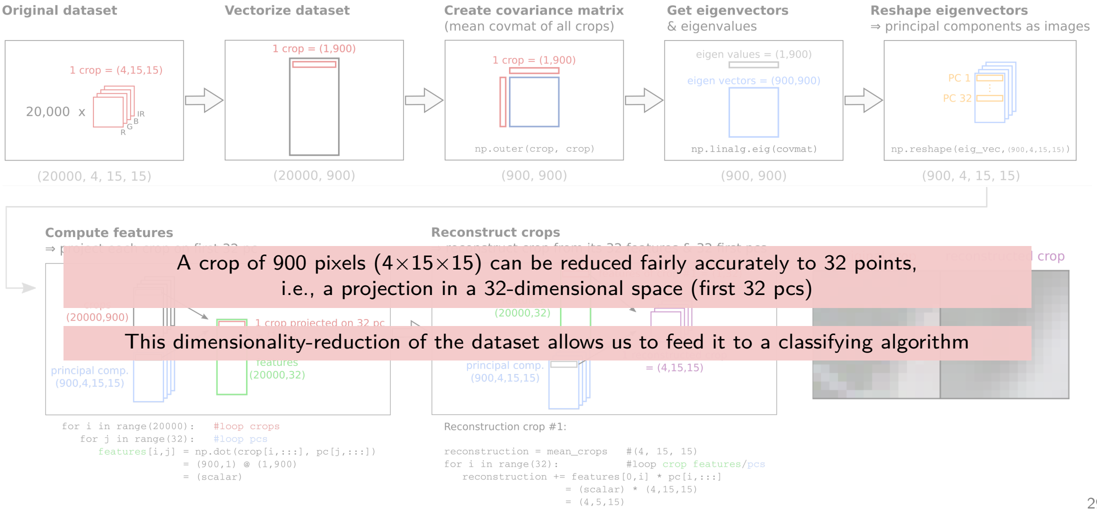


Sentinel-2 example ⇒ apply PCA on satellite image crops



A crop of 900 pixels (4×15×15) can be reduced fairly accurately to 32 points, i.e., a projection in a 32-dimensional space (first 32 pcs)

Sentinel-2 example ⇒ apply PCA on satellite image crops



A crop of 900 pixels (4×15×15) can be reduced fairly accurately to 32 points, i.e., a projection in a 32-dimensional space (first 32 pcs)

This dimensionality-reduction of the dataset allows us to feed it to a classifying algorithm

Classifying algorithms?

- **k-Nearest Neighbor (kNN)** *(last week lecture)*
  $\Rightarrow$ label images by comparing them to (annotated) images from the training set
  $\Rightarrow$ disadvantages:
  - classifier needs to keep *all* training data for future comparisons with the test data
    $\rightarrow$ *inefficient when datasets become very large ($\geq$GB)*
  - classifying a test image is expensive since it requires a comparison to all training images

- **Support Vector Machines** *(this week lecture)*
  $\Rightarrow$ parametric linear classification method
  $\Rightarrow$ advantages:
  - once the parameters are learnt, training data can be discarded
  - classification (prediction) for a new test image is fast $\rightarrow$ simple matrix multiplication with learned weights, not an exhaustive comparison to every single training data

- **Convolutional Neural Networks** *(coming weeks)*
  $\Rightarrow$ CNNs map image pixels to classes, but the mapping is more complex and will contain more parameters
  $\Rightarrow$ advantages: very powerful
  $\Rightarrow$ disadvantages: needs LOTS of data!

Classifying algorithms?

- **k-Nearest Neighbor (kNN)** *(last week lecture)*
  $\Rightarrow$ label images by comparing them to (annotated) images from the training set
  $\Rightarrow$ disadvantages:
  - classifier needs to keep *all* training data for future comparisons with the test data
    $\rightarrow$ *inefficient when datasets become very large ($\geq$GB)*
  - classifying a test image is expensive since it requires a comparison to all training images
- **Support Vector Machines** *(this week lecture)*
  $\Rightarrow$ parametric linear classification method
  $\Rightarrow$ advantages:
  - once the parameters are learnt, training data can be discarded
  - classification (prediction) for a new test image is fast $\rightarrow$ simple matrix multiplication with learned weights, not an exhaustive comparison to every single training data
- **Convolutional Neural Networks** *(coming weeks)*
  $\Rightarrow$ CNNs map image pixels to classes, but the mapping is more complex and will contain more parameters
  $\Rightarrow$ advantages: very powerful
  $\Rightarrow$ disadvantages: needs LOTS of data!

Classifying algorithms?

- **k-Nearest Neighbor (kNN)** *(last week lecture)*
  $\Rightarrow$ label images by comparing them to (annotated) images from the training set
  $\Rightarrow$ disadvantages:
  - classifier needs to keep *all* training data for future comparisons with the test data
    $\rightarrow$ *inefficient when datasets become very large ($\geq$GB)*
  - classifying a test image is expensive since it requires a comparison to all training images
- **Support Vector Machines** *(this week lecture)*
  $\Rightarrow$ parametric linear classification method
  $\Rightarrow$ advantages:
  - once the parameters are learnt, training data can be discarded
  - classification (prediction) for a new test image is fast $\rightarrow$ simple matrix multiplication with learned weights, not an exhaustive comparison to every single training data
- **Convolutional Neural Networks** *(coming weeks)*
  $\Rightarrow$ CNNs map image pixels to classes, but the mapping is more complex and will contain more parameters
  $\Rightarrow$ advantages: very powerful
  $\Rightarrow$ disadvantages: needs LOTS of data!

Classifying algorithms?

- **k-Nearest Neighbor (kNN)** *(last week lecture)*
  $\Rightarrow$ label images by comparing them to (annotated) images from the training set
  $\Rightarrow$ disadvantages:
  - classifier needs to keep *all* training data for future comparisons with the test data
    $\rightarrow$ *inefficient when datasets become very large ($\geq$GB)*
  - classifying a test image is expensive since it requires a comparison to all training images

- **Support Vector Machines** *(this week lecture)*
  $\Rightarrow$ parametric linear classification method
  $\Rightarrow$ advantages:
  - once the parameters are learnt, training data can be discarded
  - classification (prediction) for a new test image is fast $\rightarrow$ simple matrix multiplication with learned weights, not an exhaustive comparison to every single training data

- **Convolutional Neural Networks** *(coming weeks)*
  $\Rightarrow$ CNNs map image pixels to classes, but the mapping is more complex and will contain more parameters
  $\Rightarrow$ advantages: very powerful
  $\Rightarrow$ disadvantages: needs LOTS of data!

Toy example (courtesy of Andreas Ley & Ronny Hänsch)

Recap from Lecture 08

- **Task**:
  $\Rightarrow$ classify fruit images into either bananas or apples

Toy example (courtesy of Andreas Ley & Ronny Hänsch)

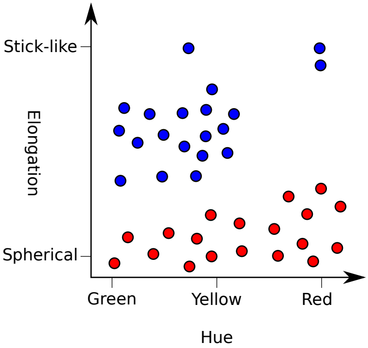
Recap from Lecture 08

- **Task**:
  ⇒ classify fruit images into either bananas or apples
- **Features (hand-crafted)**:
  ⇒ Hue (yellow to red) & Elongation (max/min extent)

Toy example (courtesy of Andreas Ley & Ronny Hänsch)

Recap from Lecture 08

- **Task**:
  $\Rightarrow$ classify fruit images into either bananas or apples
- **Features (hand-crafted)**:
  $\Rightarrow$ Hue (yellow to red) & Elongation (max/min extent)
  $\Rightarrow$ representation of input data in 2D feature space

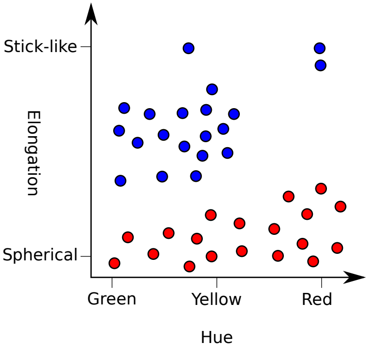


Toy example (courtesy of Andreas Ley & Ronny Hänsch)

Recap from Lecture 08

- **Task**:
  ⇒ classify fruit images into either bananas or apples

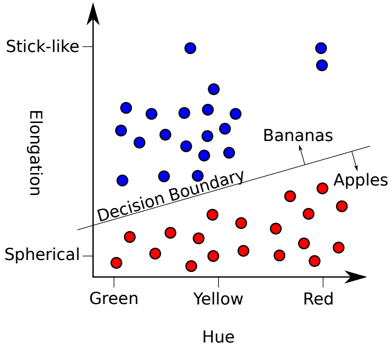
- **Features (hand-crafted)**:
  ⇒ Hue (yellow to red) & Elongation (max/min extent)
  ⇒ representation of input data in 2D feature space
  ⇒ can we "learn" which part of the feature space is bananas/apples?



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

Recap from Lecture 08

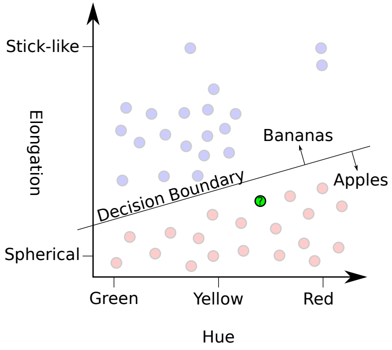
- **Task**:
  $\Rightarrow$ classify fruit images into either bananas or apples
- **Features (hand-crafted)**:
  $\Rightarrow$ Hue (yellow to red) & Elongation (max/min extent)
  $\Rightarrow$ representation of input data in 2D feature space
  $\Rightarrow$ can we "learn" which part of the feature space is bananas/apples?
- **Learning algorithm**:
  $\Rightarrow$ simple idea: split feature space into two half spaces



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

Recap from Lecture 08

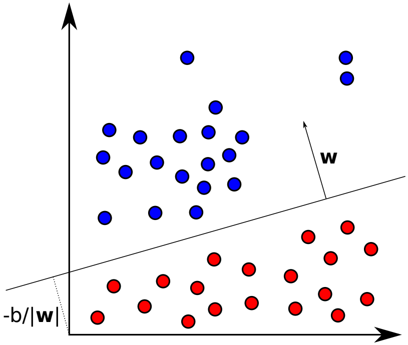
- **Task**:
  ⇒ classify fruit images into either bananas or apples
- **Features (hand-crafted)**:
  ⇒ Hue (yellow to red) & Elongation (max/min extent)
  ⇒ representation of input data in 2D feature space
  ⇒ can we "learn" which part of the feature space is bananas/apples?
- **Learning algorithm**:
  ⇒ simple idea: split feature space into two half spaces
  ⇒ classify data based on linear decision boundary



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

Recap from Lecture 08

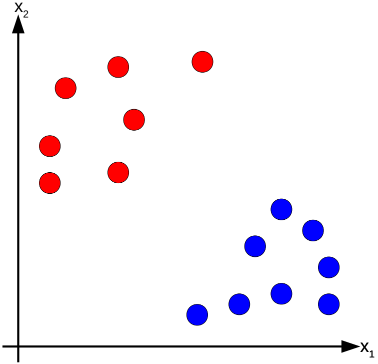
- **Task**:
  ⇒ classify fruit images into either bananas or apples
- **Features (hand-crafted)**:
  ⇒ Hue (yellow to red) & Elongation (max/min extent)
  ⇒ representation of input data in 2D feature space
  ⇒ can we "learn" which part of the feature space is bananas/apples?
- **Learning algorithm**:
  ⇒ simple idea: split feature space into two half spaces
  ⇒ classify data based on linear decision boundary
  ⇒ **perceptron**: $\boxed{y = \text{sign}(\mathbf{w}^T \mathbf{x} + b)}$
  - $y \in \{-1, 1\}$: predicted class → *banana or apple*
  - $\mathbf{x} \in \mathbb{R}^2$: feature vector → *[hue, elongation]*
  - $\mathbf{w} \in \mathbb{R}^2$: "weight vector" → *needs to be learned*
  - $b \in \mathbb{R}$: "bias" → *needs to be learned*
  - *sign*: sign function returning the sign of a real number



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

perceptron: $\boxed{y = \text{sign}(\mathbf{w}^T \mathbf{x} + b)}$

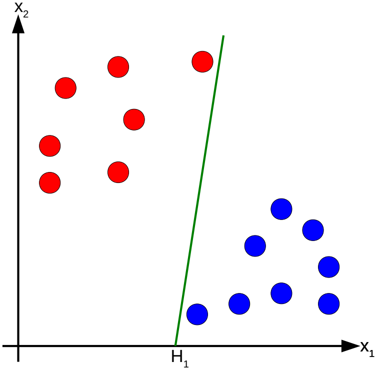
- **Best decision boundary** (**hyperplane**)?
  $\Rightarrow$ multiple "good" boundaries



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

perceptron: $\boxed{y = \text{sign}(\mathbf{w}^T \mathbf{x} + b)}$

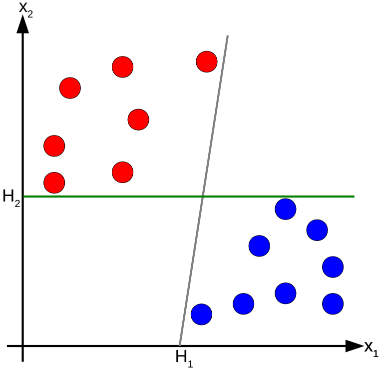
- **Best decision boundary** (**hyperplane**)?
  $\Rightarrow$ multiple "good" boundaries



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

perceptron: $\boxed{y = \mathrm{sign}(\mathbf{w}^T\mathbf{x} + b)}$

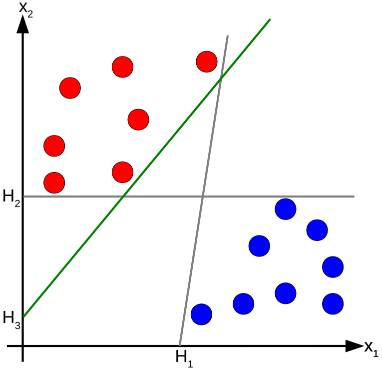
- **Best decision boundary** (**hyperplane**)?
  $\Rightarrow$ multiple "good" boundaries



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

perceptron: $\boxed{y = \text{sign}(\mathbf{w}^T \mathbf{x} + b)}$

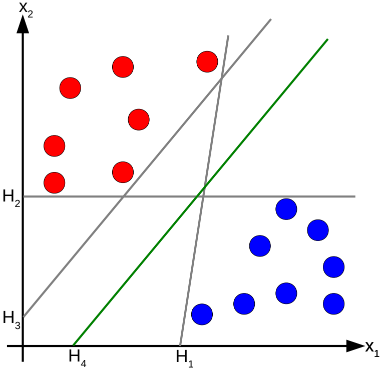
- **Best decision boundary** (**hyperplane**)?
  $\Rightarrow$ multiple "good" boundaries



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

perceptron: $\boxed{y = \text{sign}(\mathbf{w}^T\mathbf{x} + b)}$

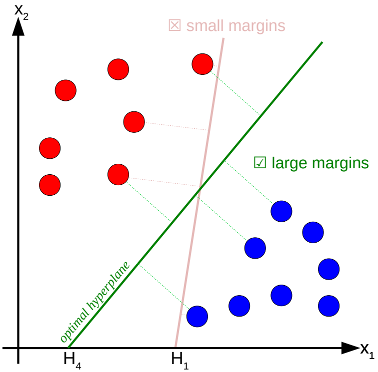
- **Best decision boundary** (**hyperplane**)?
  $\Rightarrow$ multiple "good" boundaries



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

perceptron: $\boxed{y = \text{sign}(\mathbf{w}^T \mathbf{x} + b)}$

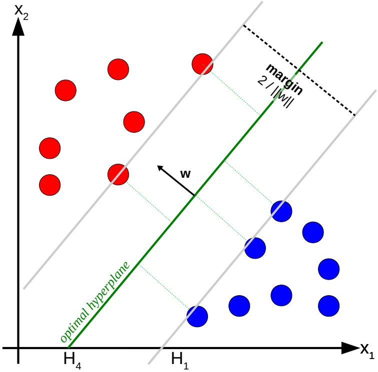
- **Best decision boundary** (**hyperplane**)?
  $\Rightarrow$ multiple "good" boundaries
  $\Rightarrow$ optimal hyperplane
  = boundary with the *maximal margin*
  = perceptron of maximal stability to new inputs



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

perceptron: $\boxed{y = \text{sign}(\mathbf{w}^T \mathbf{x} + b)}$

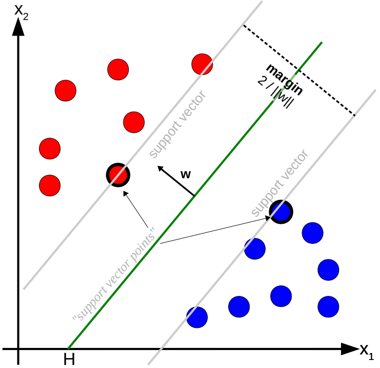
- **Best decision boundary** (**hyperplane**)?
  $\Rightarrow$ multiple "good" boundaries
  $\Rightarrow$ optimal hyperplane
  $=$ boundary with the *maximal margin*
  $=$ perceptron of maximal stability to new inputs
  $\Rightarrow$ margin $= \frac{2}{||w||}$



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

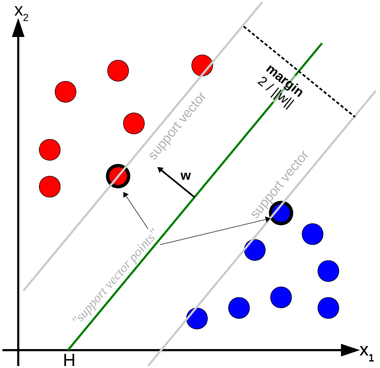
perceptron: $\boxed{y = \text{sign}(\mathbf{w}^T\mathbf{x} + b)}$

- **Best decision boundary** (**hyperplane**)?
  $\Rightarrow$ multiple "good" boundaries
  $\Rightarrow$ optimal hyperplane
  $=$ boundary with the *maximal margin*
  $=$ perceptron of maximal stability to new inputs
  $\Rightarrow$ margin $= \frac{2}{||w||}$
  $\Rightarrow$ support vector points $=$ points closest to the hyperplane
  (only these points are contributing to the result, other points are not)



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

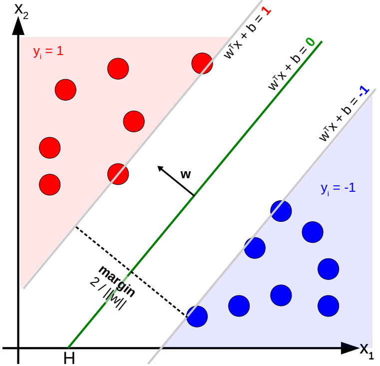
- **How can this best boundary be "learned"**?
  i.e. learn the linear classifier parameters (**w**, b)



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

- **How can this best boundary be "learned"?**
  i.e. learn the linear classifier parameters (**w**, b)
  $\Rightarrow$ maximize margin $\frac{2}{||w||}$

$$\Leftrightarrow \max_{w} \frac{2}{||w||}, \text{ subject to } \begin{cases} w^T \mathbf{x}_i + b \geq 1 & \text{if } y_i = +1 \\ w^T \mathbf{x}_i + b \leq 1 & \text{if } y_i = -1 \end{cases}$$



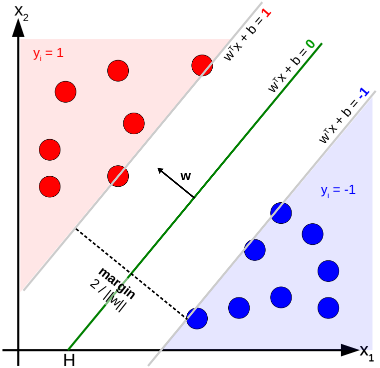
Toy example (courtesy of Andreas Ley & Ronny Hänsch)

- **How can this best boundary be "learned"**?
  i.e. learn the linear classifier parameters (**w**, b)
  $\Rightarrow$ maximize margin $\frac{2}{||w||}$

  $$\Leftrightarrow \max_{w} \frac{2}{||w||}, \text{ subject to } \begin{cases} w^T\mathbf{x}_i + b \geq 1 & \text{if } y_i = +1 \\ w^T\mathbf{x}_i + b \leq 1 & \text{if } y_i = -1 \end{cases}$$

  which is equivalent to:

  $$\Leftrightarrow \min_{w} ||w||^2, \text{ subject to } y_i(w^T\mathbf{x}_i - b) \geq 1$$



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

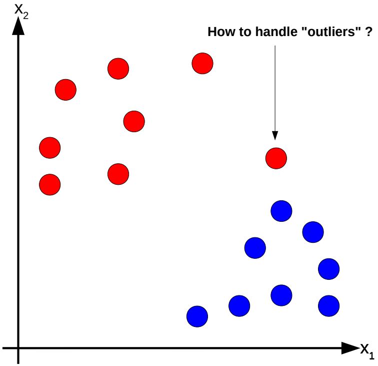
- **How can this best boundary be "learned"**?
  i.e. learn the linear classifier parameters ($\mathbf{w}$, b)
  $\Rightarrow$ maximize margin $\frac{2}{||w||}$

  $$\Leftrightarrow \max_{w} \frac{2}{||w||}, \text{ subject to } \begin{cases} w^T\mathbf{x}_i + b \geq 1 & \text{if } y_i = +1 \\ w^T\mathbf{x}_i + b \leq 1 & \text{if } y_i = -1 \end{cases}$$

  which is equivalent to:

  $$\Leftrightarrow \min_{w} ||w||^2, \text{ subject to } y_i(w^T\mathbf{x}_i - b) \geq 1$$

- **How can outliers be handled**?



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

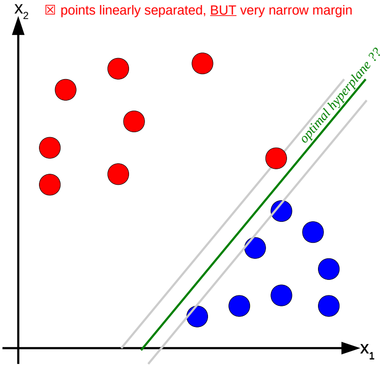
- **How can this best boundary be "learned"**?
  i.e. learn the linear classifier parameters (**w**, b)
  $\Rightarrow$ maximize margin $\frac{2}{||w||}$

  $$\Leftrightarrow \max_{w} \frac{2}{||w||}, \text{ subject to } \begin{cases} w^T \mathbf{x}_i + b \geq 1 & \text{if } y_i = +1 \\ w^T \mathbf{x}_i + b \leq 1 & \text{if } y_i = -1 \end{cases}$$

  which is equivalent to:

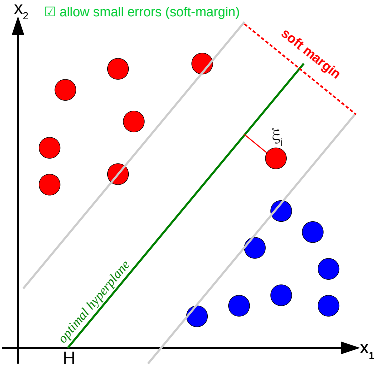
  $$\Leftrightarrow \min_{w} ||w||^2, \text{ subject to } y_i(w^T \mathbf{x}_i - b) \geq 1$$

- **How can outliers be handled**?
  $\Rightarrow$ is a hard-margin with 100% accuracy good?



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

- **How can this best boundary be "learned"**?
  i.e. learn the linear classifier parameters (**w**, b)
  $\Rightarrow$ maximize margin $\frac{2}{||w||}$
  $$\Leftrightarrow \max_{w} \frac{2}{||w||}, \text{ subject to } \begin{cases} w^T\mathbf{x}_i + b \geq 1 & \text{if } y_i = +1 \\ w^T\mathbf{x}_i + b \leq 1 & \text{if } y_i = -1 \end{cases}$$
  which is equivalent to:
  $$\Leftrightarrow \min_{w} ||w||^2, \text{ subject to } y_i(w^T\mathbf{x}_i - b) \geq 1$$
- **How can outliers be handled**?
  $\Rightarrow$ is a hard-margin with 100% accuracy good?
  $\Rightarrow$ no, allow small errors to favour overall better model
  $\Leftrightarrow$ favour large margin boundaries
  $\Leftrightarrow$ tolerate margin violation (**soft-margin**)



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

- **How can this best boundary be "learned"**?
  i.e. learn the linear classifier parameters (**w**, b)
  $\Rightarrow$ maximize margin $\frac{2}{||w||}$

  $$\Leftrightarrow \max_{w} \frac{2}{||w||}, \text{ subject to } \begin{cases} w^T \mathbf{x}_i + b \geq 1 & \text{if } y_i = +1 \\ w^T \mathbf{x}_i + b \leq 1 & \text{if } y_i = -1 \end{cases}$$

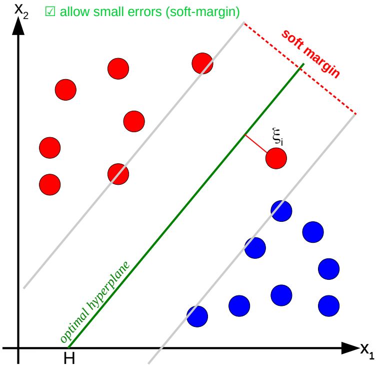
  which is equivalent to:

  $$\Leftrightarrow \min_{w} ||w||^2, \text{ subject to } y_i(w^T \mathbf{x}_i - b) \geq 1$$

- **How can outliers be handled**?
  $\Rightarrow$ is a hard-margin with 100% accuracy good?
  $\Rightarrow$ no, allow small errors to favour overall better model
  $\Leftrightarrow$ favour large margin boundaries
  $\Leftrightarrow$ tolerate margin violation (**soft-margin**)
  $\Rightarrow$ optimization becomes:

  $$\min_{w,\xi_i} ||w||^2 + C \sum_{i}^{N} \xi_i, \text{ subject to } y_i(w^T \mathbf{x}_i - b) \geq 1 - \xi_i$$

  where C is a regularization parameter:
  small C $\Rightarrow$ constraints easily ignored $\Rightarrow$ large margin; large C $\Rightarrow$ opposite



Side note: *reformulating optimization in terms of* ***regularization*** *and* ***loss function*** *(anticipating DL lectures)*

Learning an SVM has been formulated as a *constrained* optimization problem over w and ξ:

$$\min_{w,\xi_i} ||w||^2 + C\sum_i^N \xi_i \qquad \text{subject to:} \quad y_i(w^T\mathbf{x}_i - b) \geq 1 - \xi_i$$

The constraint $y_i(w^T\mathbf{x}_i - b) \geq 1 - \xi_i$ can be written more concisely as: $y_i f(\mathbf{x}_i) \geq 1 - \xi_i$

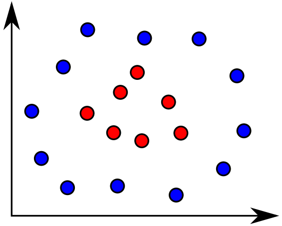
Together with $\xi_i > 0$, it is equivalent to: $\xi_i = max(0, 1 - y_i f(x_i))$

Hence the learning problem is equivalent to the *unconstrained* optimization problem over w:

$$\min_{w} \underbrace{||w||^2}_{\text{regularization}} + C\sum_i^N \underbrace{max(0,\ 1 - y_i f(x_i))}_{\text{loss function (Hinge loss)}}$$

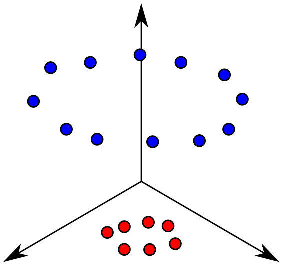
Toy example (courtesy of Andreas Ley & Ronny Hänsch)

- **What if the features x_i are not linearly separable**?



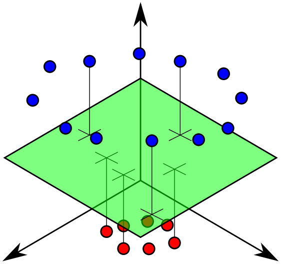
Toy example (courtesy of Andreas Ley & Ronny Hänsch)

- **What if the features x_i are not linearly separable**?
  $\Rightarrow$ compute new features $x_i \mapsto \phi(x)$
  $\phi(x)$ is a **feature map**, mapping x to $\phi(x)$ where data is separable



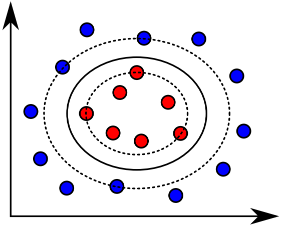
Toy example (courtesy of Andreas Ley & Ronny Hänsch)

- **What if the features x_i are not linearly separable?**
  $\Rightarrow$ compute new features $x_i \mapsto \phi(x)$
  $\phi(x)$ is a **feature map**, mapping x to $\phi(x)$ where data is separable
  $\Rightarrow$ solve for $\mathbf{w}$ in high dimensional feature space



Toy example (courtesy of Andreas Ley & Ronny Hänsch)

- **What if the features x_i are not linearly separable**?
  $\Rightarrow$ compute new features $x_i \mapsto \phi(x)$
  $\phi(x)$ is a **feature map**, mapping x to $\phi(x)$ where data is separable
  $\Rightarrow$ solve for **w** in high dimensional feature space
  $\Rightarrow$ data not lineary-seperable in original feature space become separable



Kernel trick

The *Representer Theorem* states that the solution **w** can be written as a linear combination of the training data:

$$w = \sum_{j=1}^{N} \alpha_j y_j x$$

Kernel trick

The *Representer Theorem* states that the solution **w** can be written as a linear combination of the training data:

$$w = \sum_{j=1}^{N} \alpha_j y_j x$$

The linear classifier can therefore be reformulated as:

$$\begin{aligned} f(x) &= w^T x + b \\ &= \sum_{i}^{N} \alpha_i y_i (x_i^T x) + b \end{aligned}$$

Kernel trick

The *Representer Theorem* states that the solution **w** can be written as a linear combination of the training data:

$$w = \sum_{j=1}^{N} \alpha_j y_j x$$

The linear classifier can therefore be reformulated as:

$$\begin{aligned} f(x) &= w^T x + b \\ &= \sum_{i}^{N} \alpha_i y_i (x_i^T x) + b \end{aligned}$$

NB: this reformulation seems to have the disadvantage of a K-NN classifier, i.e. requires the training data points x_i. However, many of the $\alpha_i = 0$: the ones that are non-zero define the support vector points x_i

Kernel trick

The *Representer Theorem* states that the solution **w** can be written as a linear combination of the training data:

$$w = \sum_{j=1}^{N} \alpha_j y_j x$$

The linear classifier can therefore be reformulated as:

$$\begin{aligned} f(x) &= w^T x + b \\ &= \sum_{i}^{N} \alpha_i y_i (x_i^T x) + b \end{aligned}$$

NB: this reformulation seems to have the disadvantage of a K-NN classifier, i.e. requires the training data points x_i. However, many of the $\alpha_i = 0$: the ones that are non-zero define the support vector points x_i

Using the feature map $\phi(x)$, it can be reformulated as:

$$\begin{aligned} f(x) &= \sum_{i}^{N} \alpha_i y_i (\phi(x_i)^T \phi(x)) + b \\ &= \sum_{i}^{N} \alpha_i y_i k(x_i, x) + b \end{aligned}$$

where $k(x_i, x)$ is known as a **Kernel**

Kernel trick

- Classifier can be learnt and applied without explicitly computing $\phi(x)$
- All that is required is the kernel $k(x, x')$
- Multiple kernels exist:
  - linear kernels: $k(x, x') = x^T x'$
    $\rightarrow$ *very fast and easy to train, but very simple*
  - polynomial kernels: $k(x, x') = (1 + x^T x')^d$
    $\rightarrow$ *contains all polynomial terms up to degree d*
  - gaussian kernels: $k(x, x') = exp(-||x - x'||^2 / 2\sigma^2)$ *(RBF kernel)*
    $\rightarrow$ *kernel very powerful and most often used*

Kernel trick

- Classifier can be learnt and applied without explicitly computing $\phi(x)$
- All that is required is the kernel $k(x, x')$
- Multiple kernels exist:
  - linear kernels: $k(x, x') = x^T x'$
    $\rightarrow$ *very fast and easy to train, but very simple*
  - polynomial kernels: $k(x, x') = (1 + x^T x')^d$
    $\rightarrow$ *contains all polynomial terms up to degree d*
  - gaussian kernels: $k(x, x') = exp(-||x - x'||^2/2\sigma^2)$ *(RBF kernel)*
    $\rightarrow$ *kernel very powerful and most often used*

Kernel trick

- Classifier can be learnt and applied without explicitly computing $\phi(x)$
- All that is required is the kernel $k(x, x')$
- Multiple kernels exist:
  - linear kernels: $k(x, x') = x^T x'$
    $\rightarrow$ *very fast and easy to train, but very simple*
  - polynomial kernels: $k(x, x') = (1 + x^T x')^d$
    $\rightarrow$ *contains all polynomial terms up to degree d*
  - gaussian kernels: $k(x, x') = exp(-||x - x'||^2 / 2\sigma^2)$ *(RBF kernel)*
    $\rightarrow$ *kernel very powerful and most often used*

1. **HOG features + SVM for object detection**
   - Original idea: *Dalal and Triggs (2005) - "Histograms of Oriented Gradients for Human Detection"*
   - Adaptation: *detect ships in satellite imagery* (notebook using skimage/sklearn)

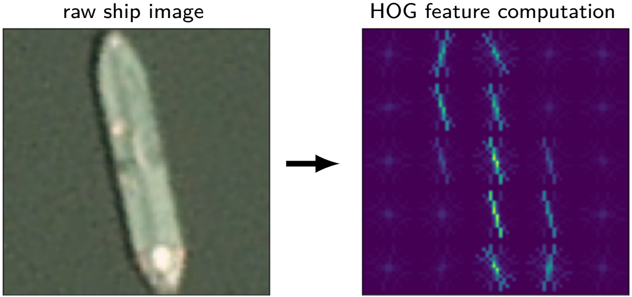
1. **HOG features + SVM for object detection**
   - Original idea: *Dalal and Triggs (2005) - "Histograms of Oriented Gradients for Human Detection"*
   - Adaptation: *detect ships in satellite imagery* (notebook using skimage/sklearn)

1. **HOG features + SVM for object detection**
   - Original idea: *Dalal and Triggs (2005) - "Histograms of Oriented Gradients for Human Detection"*
   - Adaptation: *detect ships in satellite imagery* (notebook using skimage/sklearn)

raw ship image

1. **HOG features + SVM for object detection**

- Original idea: *Dalal and Triggs (2005) - "Histograms of Oriented Gradients for Human Detection"*
- Adaptation: *detect ships in satellite imagery* (notebook using skimage/sklearn)

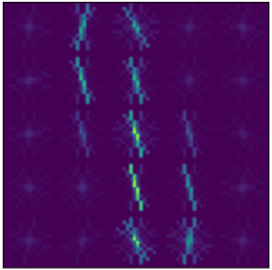


1. **HOG features + SVM for object detection**

- Original idea: *Dalal and Triggs (2005) - "Histograms of Oriented Gradients for Human Detection"*
- Adaptation: *detect ships in satellite imagery* (notebook using skimage/sklearn)

raw ship image

HOG feature computation



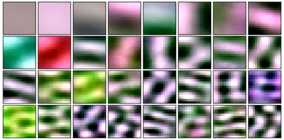
train SVM for classification

```
Accuracy: 0.99625

              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       589
         1.0       1.00      0.99      0.99       211

    accuracy                           1.00       800
   macro avg       1.00      0.99      1.00       800
weighted avg       1.00      1.00      1.00       800
```

2. **Classify land use in satellite images (Sentinel-2)**

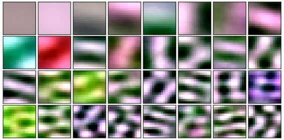
2. **Classify land use in satellite images (Sentinel-2)**

PCA dimensionality reduction



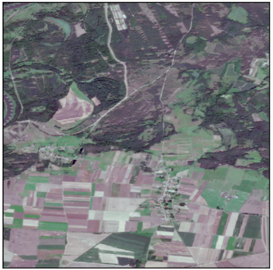
2. **Classify land use in satellite images (Sentinel-2)**

PCA dimensionality reduction



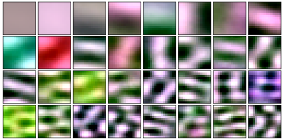
→

train SVM & apply

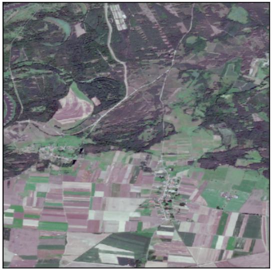


2. **Classify land use in satellite images (Sentinel-2)**

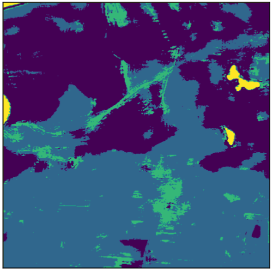
PCA dimensionality reduction



train SVM & apply

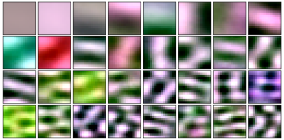


land-use classification

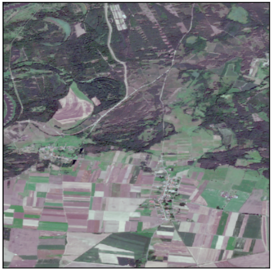


2. **Classify land use in satellite images (Sentinel-2)**

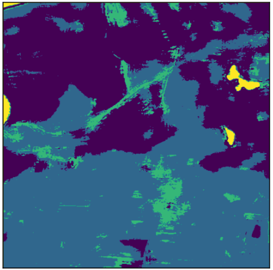
PCA dimensionality reduction



train SVM & apply



land-use classification



EXERCISE !