CS5489 - Machine Learning

Lecture 4c - Classification Summary

Prof. Antoni B. Chan

Dept. of Computer Science, City University of Hong Kong

Outline

1. Nonlinear classifiers
2. Kernel trick and kernel SVM
3. Ensemble Methods - Boosting, Random Forests
4. **Classification Summary**

Feature Pre-processing

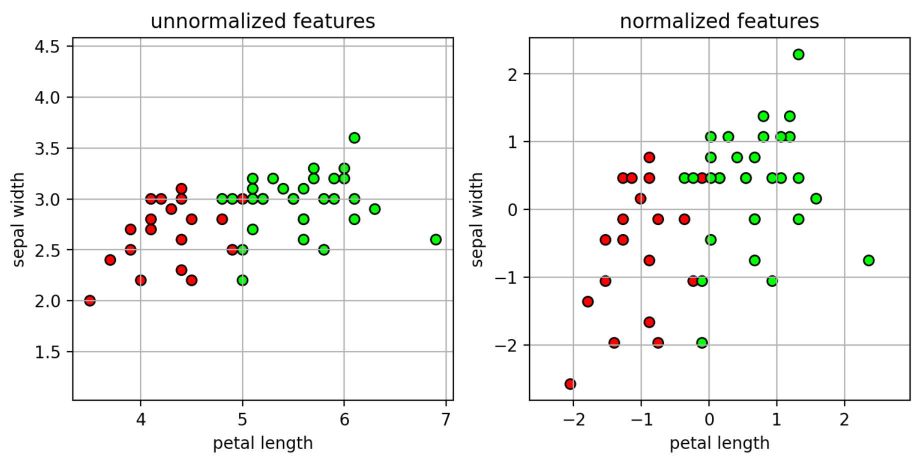
- Some classifiers, such as SVM and LR, are sensitive to the scale of the feature values.
  - feature dimensions with larger values may dominate the objective function.
- Common practice is to *standardize* or *normalize* each feature dimension before learning the classifier.
  - Two Methods...

- **Method 1:** scale each feature dimension so the mean is 0 and variance is 1.
  - $\tilde{x}_d = \frac{1}{s}(x_d - m)$
  - s is the standard deviation of feature values.
  - m is the mean of the feature values.
- **NOTE:** the parameters for scaling the features should be estimated from the training set!
  - same scaling is applied to the test set.

In [5]:
```
# using the iris data
scaler = preprocessing.StandardScaler()  # make scaling object
trainXn = scaler.fit_transform(trainX)   # use training data to fit scaling parameters
testXn  = scaler.transform(testX)        # apply scaling to test data
```

In [7]:
```
nfig1
```

Out[7]:

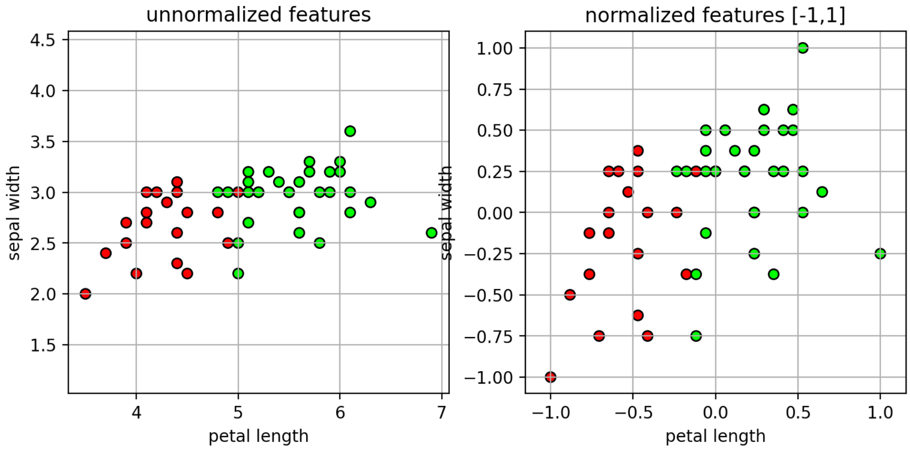


- **Method 2:** scale features to a fixed range, -1 to 1.
  - $\tilde{x}_d = 2 * (x_d - min)/(max - min) - 1$
  - max and min are the maximum and minimum features values.

In [8]:
```
# using the iris data
scaler = preprocessing.MinMaxScaler(feature_range=(-1,1))     # make scaling object
trainXn = scaler.fit_transform(trainX)   # use training data to fit scaling parameters
testXn  = scaler.transform(testX)         # apply scaling to test data
```

In [10]:
```
nfig2
```

Out[10]:



Data Representation and Feature Engineering

- How to represent data as a vector of numbers?
  - the encoding of the data into a feature vector should make sense
  - inner-products or distances calculated between feature vectors should be meaningful in terms of the data.

- Categorical variables
  - Example: x has 3 possible category labels: cat, dog, horse
  - We could encode this as: $x = 0$, $x = 1$, and $x = 2$.
    - Suppose we have two data points: $x = cat$, $x' = horse$.
    - What is the meaning of $x * x' = 2$?

One-hot encoding

- encode a categorical variable as a vector of ones and zeros
  - if there are K categories, then the vector is K dimensions.
- Example:
  - x=cat $\rightarrow$ x=[1 0 0]
  - x=dog $\rightarrow$ x=[0 1 0]
  - x=horse $\rightarrow$ x=[0 0 1]

In [11]:
```
# one-hot encoding example
X = [['cat'], ['dog'], ['cat'], ['bird'], ['dog']]  # each row is a sample
ohe = preprocessing.OneHotEncoder(sparse=False)
ohe.fit(X)          # map the categories to one-hot vectors
print(ohe.categories_)
ohe.transform(X)    # transform to one-hot-encoding
```

```
[array(['bird', 'cat', 'dog'], dtype=object)]
```

Out[11]:
```
array([[0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.]])
```

Binning

- encode a real value as a vector of ones and zeros
  - assign each feature value to a bin, and then use one-hot-encoding

In [12]:
```
# example
X = [[-3], [0.5], [1.5], [2.5]]  # the data
bins = [-2,-1,0,1,2]       # define the bin edges

# map from value to bin number
Xbins = digitize(X, bins=bins)

# map from bin number (0..5) to 0-1 vector
ohe = preprocessing.OneHotEncoder(categories=[arange(6)], sparse=False)
ohe.fit(Xbins)
ohe.transform(Xbins)
```

Out[12]:
```
array([[1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1.]])
```

Data transformations - polynomials

- Represent interactions between features using polynomials
- Example:
  - 2nd-degree polynomial models pair-wise interactions
    - $[x_1, x_2] \rightarrow [x_1^2, x_1 x_2, x_2^2]$
  - Combine with other degrees:
    - $[x_1, x_2] \rightarrow [1, x_1, x_2, x_1^2, x_1 x_2, x_2^2]$

In [13]:
```
X = [[0,1], [1,2], [3,4]]
pf = preprocessing.PolynomialFeatures(degree=2)
pf.fit(X)
pf.transform(X)
```

Out[13]:
```
array([[ 1.,  0.,  1.,  0.,  0.,  1.],
       [ 1.,  1.,  2.,  1.,  2.,  4.],
       [ 1.,  3.,  4.,  9., 12., 16.]])
```

Data transformations - univariate

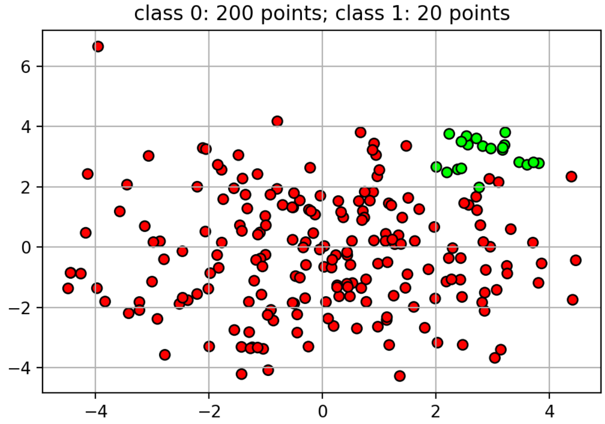
- Apply a non-linear transformation to the feature
  - e.g., x $\rightarrow$ log(x)
  - useful if the dynamic range of x is very large

Unbalanced Data

- For some classification tasks that data will be unbalanced
  - many more examples in one class than the other.
- **Example:** detecting credit card fraud
  - credit card fraud is rare
    - 50 examples of fraud, 5000 examples of legitimate transactions.

In [15]:
```
udatafig
```

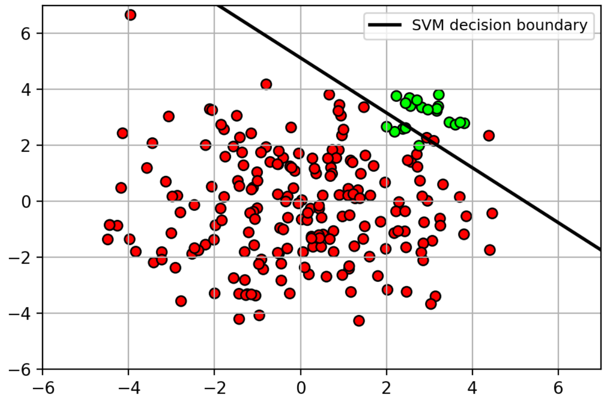
Out[15]:



- Unbalanced data can cause problems when training the classifier
  - classifier will focus more on the class with more points.
  - decision boundary is pushed away from class with more points

In [17]:
```
udatafig1
```

Out[17]:



- **Solution:** apply weights on the classes during training.
  - weights are inversely proportional to the class size.

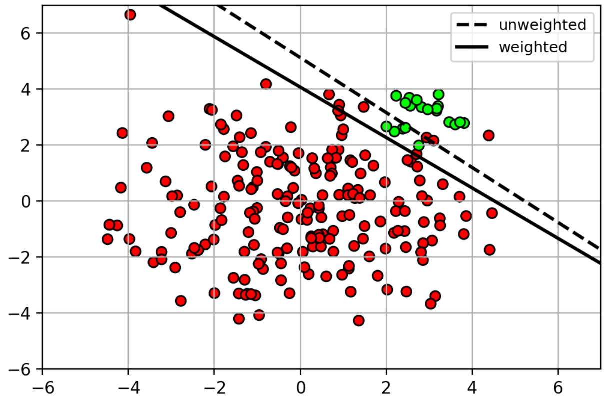
In [18]:
```
clfw = svm.SVC(kernel='linear', C=10,  class_weight='balanced')
clfw.fit(X, Y)

print("class weights =", clfw.class_weight_)
```

```
class weights = [0.55 5.5 ]
```

In [20]:
```
udatafig2
```

Out[20]:



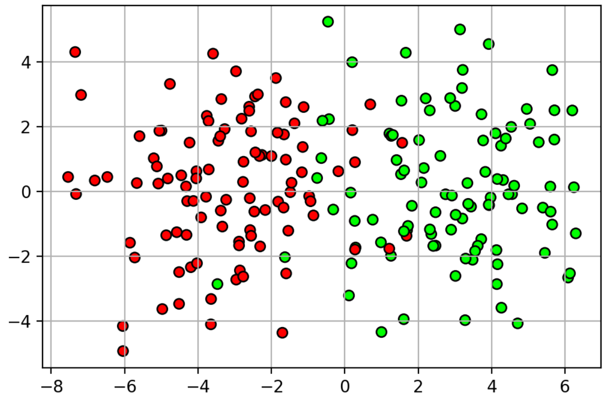
Classifier Imbalance

- In some tasks, errors on certain classes cannot be tolerated.

- **Example:** detecting spam vs non-spam
  - non-spam should *definitely not* be marked as spam
    - okay to mark some spam as non-spam

In [22]:
```
udatafig3
```

Out[22]:



...

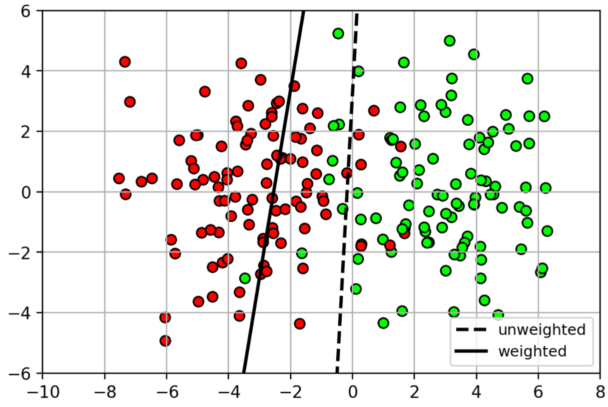
- Class weighting can be used to make the classifier focus on certain classes
  - e.g., weight non-spam class higher than spam class
    - classifier will try to correctly classify all non-spam samples, at the expense of making errors on spam samples.

In [23]:
```
# dictionary (key,value) = (class name, class weight)
cw = {0: 0.2,
      1:  5}  # class 1 is 25 times more important!

clfw = svm.SVC(kernel='linear', C=10,  class_weight=cw)
clfw.fit(X, Y);
```

In [25]:
```
udatafig4
```

Out[25]:



...

Classification Summary

- **Classification task**
  - Observation $\mathbf{x}$: typically a real vector of feature values, $\mathbf{x} \in \mathbb{R}^d$.
  - Class y: from a set of possible classes, e.g., $\mathcal{Y} = \{0, 1\}$
  - **Goal:** given an observation $\mathbf{x}$, predict its class y.

Name	Type	Classes	Decision function	Training	Advantages	Disadvantages
Bayes' classifier	generative	multi-class	non-linear	estimate class-conditional densities $p(x\|y)$ by maximizing likelihood of data.	- works well with small amounts of data. - multi-class.	- depends on the data correctly fitting the class-conditional.

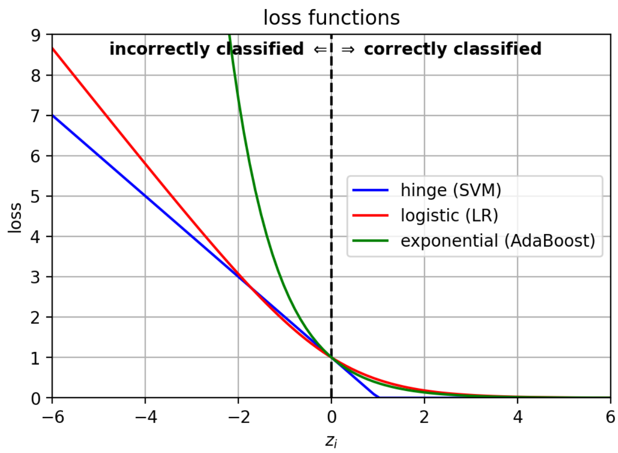
					- minimum probability of error if probability models are correct.	
logistic regression	discriminative	binary	linear	maximize likelihood of data in $p(y\|x)$.	- well-calibrated probabilities. - efficient to learn.	- linear decision boundary. - sensitive to C parameter.
support vector machine (SVM)	discriminative	binary	linear	maximize the margin (distance between decision surface and closest point).	- works well in high-dimension. - good generalization.	- linear decision boundary. - sensitive to C parameter.
kernel SVM	discriminative	binary	non-linear (kernel function)	maximize the margin.	- non-linear decision boundary. - can be applied to non-vector data using appropriate kernel.	- sensitive to kernel function and hyperparameters. - high memory usage for large datasets
AdaBoost	discriminative	binary	non-linear (ensemble of weak learners)	train successive weak learners to focus on misclassified points.	- non-linear decision boundary. can do feature selection. - good generalization.	- sensitive to outliers.
XGBoost	discriminative	binary	non-linear (ensemble of decision trees)	train successive learners to focus on gradient of the loss.	- non-linear decision boundary. - good generalization.	- sensitive to outliers.
Random Forest	discriminative	multi-class	non-linear (ensemble of	aggregate predictions over several decision trees, trained using different	- non-linear decision boundary. can do feature selection. - good generalization.	- sensitive to outliers.

Loss functions

- The classifiers differ in their loss functions, which influence how they work.
  - $z_i = y_i f(\mathbf{x}_i)$

In [27]:
```
lossfig
```

Out[27]:

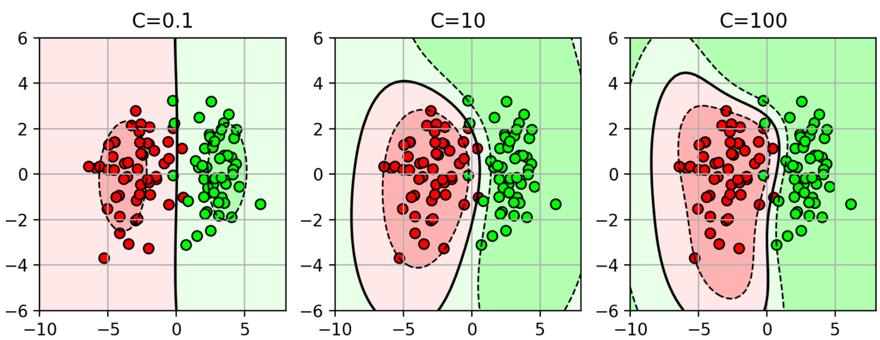


Regularization and Overfitting

- Some models have terms to prevent overfitting the training data.
  - this can improve *generalization* to new data.
- There is a parameter to control the regularization effect.
  - select this parameter using cross-validation on the training set.

In [29]:
```
ofig
```

Out[29]:



Structural Risk Minimization

- A general framework for balancing data fit and model complexity.

- Many learning problems can be written as a combination of data-fit and regularization term:

$$f^* = \operatorname*{argmin}_f \sum_i L(y_i, f(\mathbf{x}_i)) + \lambda\Omega(f)$$

- assume f within some class of funcitions, e.g., linear functions $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$.
- L is the loss function, e.g., logistic loss.
- Ω is the regularization function on f, e.g., $||\mathbf{w}||^2$
- λ is the tradeoff parameter, e.g., $1/C$.

Other things

- *Multiclass classification*
  - can use binary classifiers to do multi-class using *1-vs-rest* formulation.
- *Feature normalization*
  - normalize each feature dimension so that some feature dimensions with larger ranges do not dominate the optimization process.
- *Unbalanced data*
  - if more data in one class, then apply weights to each class to balance objectives.
- *Class imbalance*
  - mistakes on some classes are more critical.
  - reweight class to focus classifier on correctly predicting one class at the expense of others.

Applications

- Web document classification, spam classification
- Face gender recognition, face detection, digit classification

Features

- Choice of features is important!
  - using uninformative features may confuse the classifier.
  - use domain knowledge to pick the best features to extract from the data.

Which classifier is best?

- **"No Free Lunch" Theorem** (Wolpert and Macready)

> "If an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems."

- In other words, there is no *best* classifier for all tasks. The best classifier depends on the particular problem.