



Computer
Science

CSC380: Principles of Data Science

Predictive Modeling and Classification 2

Xinchen Yu

- Withdrawn DDL: Oct 29
- HW5 due this Friday, Oct 27 by 11:59 pm
- Final exam: will spend two lectures for the final review
 - We will reuse some midterm problems with a bit of variations.

Outline

- Decision tree variations
- KNN
- Model selections and evaluations

How to construct a tree

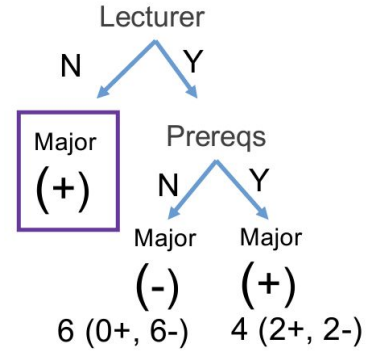
4

- Assign all training instances to the root of the tree. Set current node to root node.
- For each feature:
 - a. Partition all data instances at the node by the value of the feature.
 - b. Compute the accuracy from the partitioning.
- Identify feature that results in the highest accuracy. Set this feature to be the splitting criterion at the current node.

		Prereqs	Lecturer	HasLabs	
		↓	↓	↓	
Rating	Easy?	Alt?	Sys?	Thy?	Morning?
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y

How to construct a tree

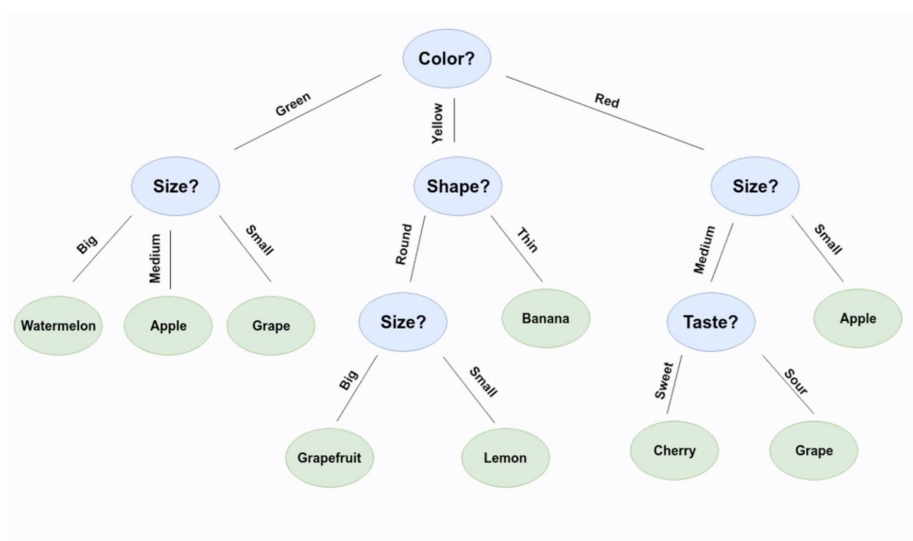
- Partition all instances according to attribute value of the best feature.
- Denote each partition as a child node of the current node.
- For each child node:
 - a. If the child node is “pure” (has instances from only one class) tag it as a leaf and return.
 - b. If not set the child node as the current node and recurse to step 2.



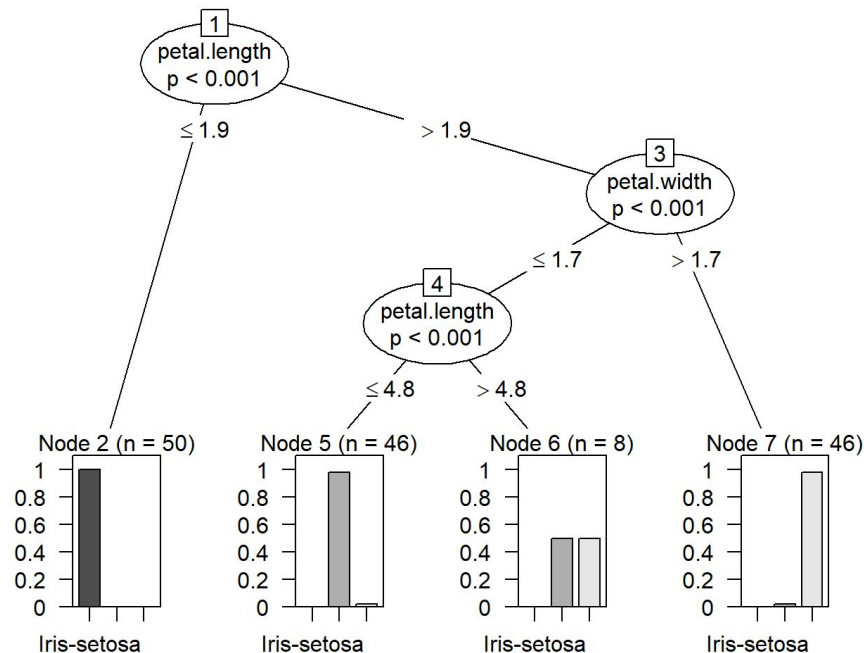
- Binary
- Categorical: values in $\{1, \dots, C\}$ e.g., occupation, blood type
 - Option 1: Instead of 2 children, have C children.
 - Option 2: Derive C features of the form “feature= c ?” for every $c \in C$.

↑ binary features!

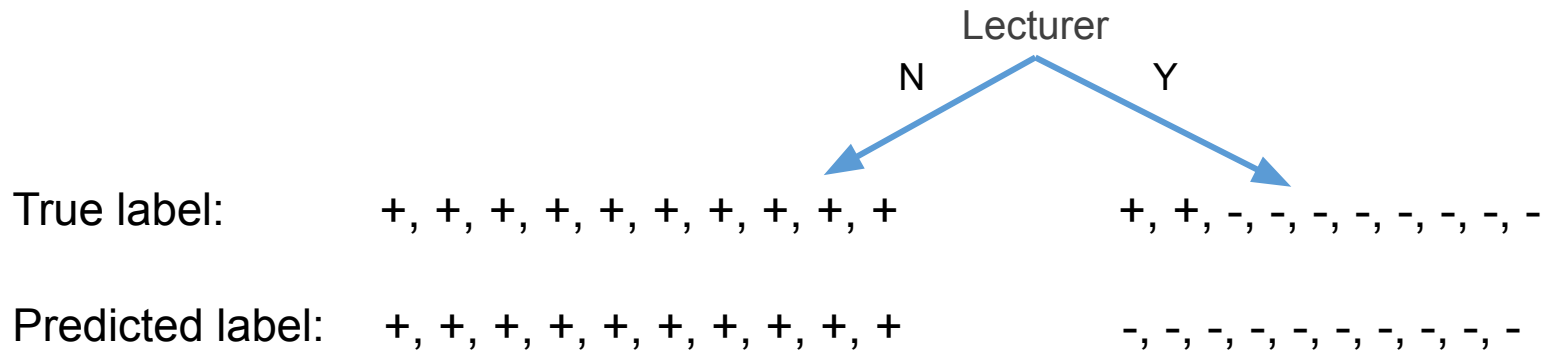
Option 1:



- Binary
- Multiclass: What changes do we need to make?
 - Almost none!
 - Just extend the accuracy to multiclass.



Review: criterion is accuracy



Accuracy: $\frac{10}{20} \cdot \frac{10}{10} + \frac{10}{20} \cdot \frac{8}{10} = \frac{18}{20} = 0.9$

- Sum of (fraction of subgroup * fraction of correct answer in subgroup)
- What if we change it to Sum of (fraction of a subgroup * some function on that subgroup)?

Different selection criteria

Notions of uncertainty: general case

Suppose in $S \subseteq \mathcal{X} \times \mathcal{Y}$, a p_k fraction are labeled as k (for each $k \in \mathcal{Y}$).

1 Classification error:

$$u(S) := 1 - \max_{k \in \mathcal{Y}} p_k$$

2 Gini index:

$$u(S) := 1 - \sum_{k \in \mathcal{Y}} p_k^2$$

3 Entropy:

$$u(S) := \sum_{k \in \mathcal{Y}} p_k \log \frac{1}{p_k}$$

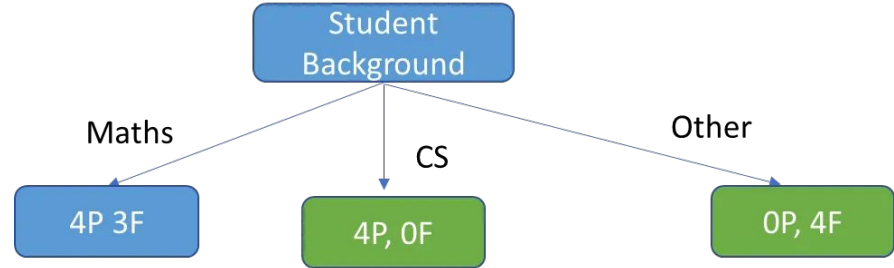
Each is *maximized* when $p_k = 1/|\mathcal{Y}|$ for all $k \in \mathcal{Y}$
(i.e., equal numbers of each label in S)

Each is *minimized* when $p_k = 1$ for a single label $k \in \mathcal{Y}$
(so S is **pure** in label)

Different selection criteria - Gini index

② Gini index:

$$u(S) := 1 - \sum_{k \in \mathcal{Y}} p_k^2$$



$$Gini_{maths} = 1 - \left(\frac{4}{7}\right)^2 - \left(\frac{3}{7}\right)^2 = .4897$$

$$Gini_{others} = 1 - \left(\frac{0}{4}\right)^2 - \left(\frac{4}{4}\right)^2 = 0$$

$$Gini_{CS} = 1 - \left(\frac{4}{4}\right)^2 - \left(\frac{0}{4}\right)^2 = 0$$

$$Gini_{bkgrd} = \frac{7}{15} * .4897 + \frac{4}{15} * 0 + \frac{4}{15} * 0 = .2286$$

Pick the one with lowest
Gini index as root

k-Nearest Neighbors (k-NN)

Error := 1 – accuracy.

Suppose we have trained a function \hat{f} on $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^m$ using a supervised learning algorithm.

- Train error: Evaluate on D .

$$\widehat{err}_D(\hat{f}) := \frac{1}{|D|} \sum_{(x,y) \in D} \mathbf{I}\{\hat{f}(x) \neq y\}$$

- Test error: Evaluate on $D' = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m'}$ not used for training.

Q: Choose one:

(1) train error \geq test error (2) train error \approx test error (3) train error \leq test error

Background: Train Error

Suppose our train set:

Index	Ground Truth	Predicted
0	0	1
1	1	1
2	0	0
3	0	1
4	1	0
5	0	0
6	1	1

$$\frac{1}{6} \cdot (1 + 0 + 0 + 1 + 1 + 0) = \frac{1}{2}$$

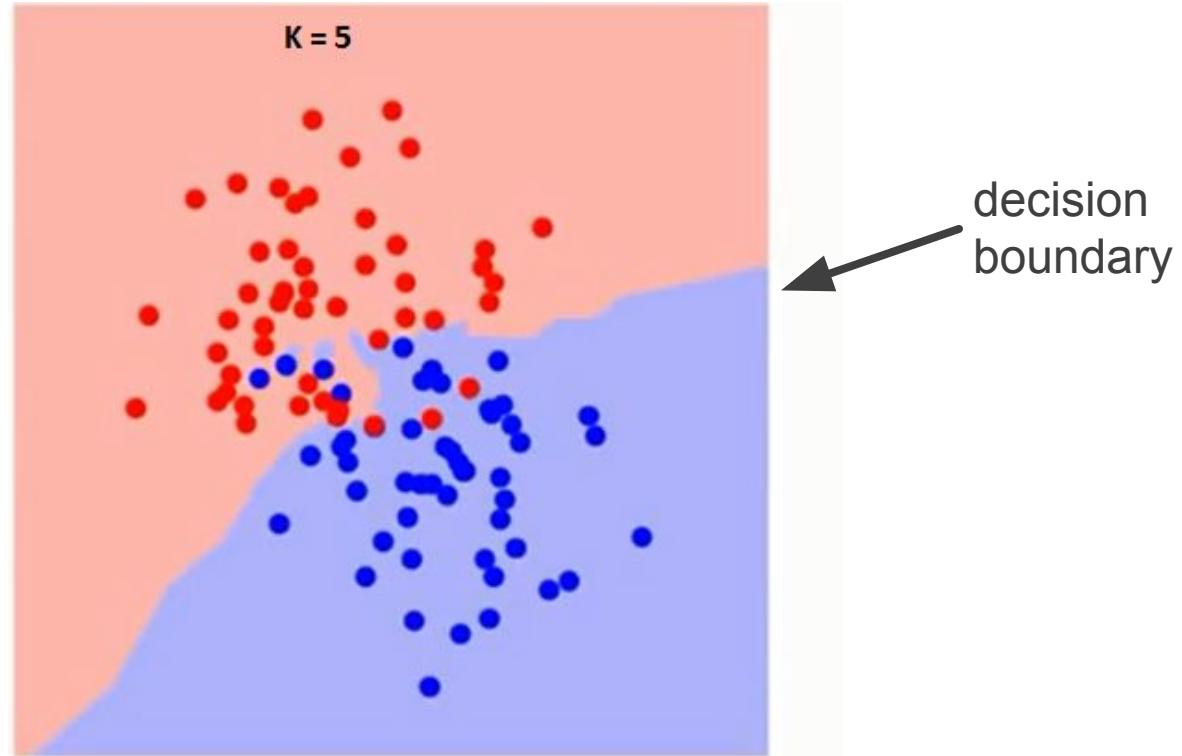
Standard practice:

- Given a data set D , split it into train set D_{train} and D_{test}
 - large data: 90-10 ratio
 - medium data: 80-20 ratio
 - small data: 70-30 ratio(these are guidelines only)
- Train on D_{train} and evaluate error rate on D_{test} . You trust that D_{test} will be the performance when you deploy the trained classifier.

- Train set: $S = \{ (x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)}) \}$
- **Idea**: given a new, unseen data point x , its label should resemble the labels of **nearby points**
- What function?
 - Input: $x \in \mathbb{R}^d$
 - From S , find the k nearest points to x from S ; call it $N(x)$
 - Output: the majority vote of $\{y_i: i \in N(x)\}$
 - For regression, take the average label.

E.g., Euclidean distance





How to extract features as **real values**?

- Binary features: Take 0/1
- Categorical $\{1, \dots, C\}$ (e.g., movie genres)
 - Binary vector of length C . Set c -th coordinate 1 and 0 otherwise. one-hot encoding

id	color
1	red
2	blue
3	green
4	blue



id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

How to extract features as **real values**?

- Binary features: Take 0/1
- Categorical $\{1, \dots, C\}$ (e.g., movie genres)
 - Binary vector of length C . Set c -th coordinate 1 and 0 otherwise. one-hot encoding

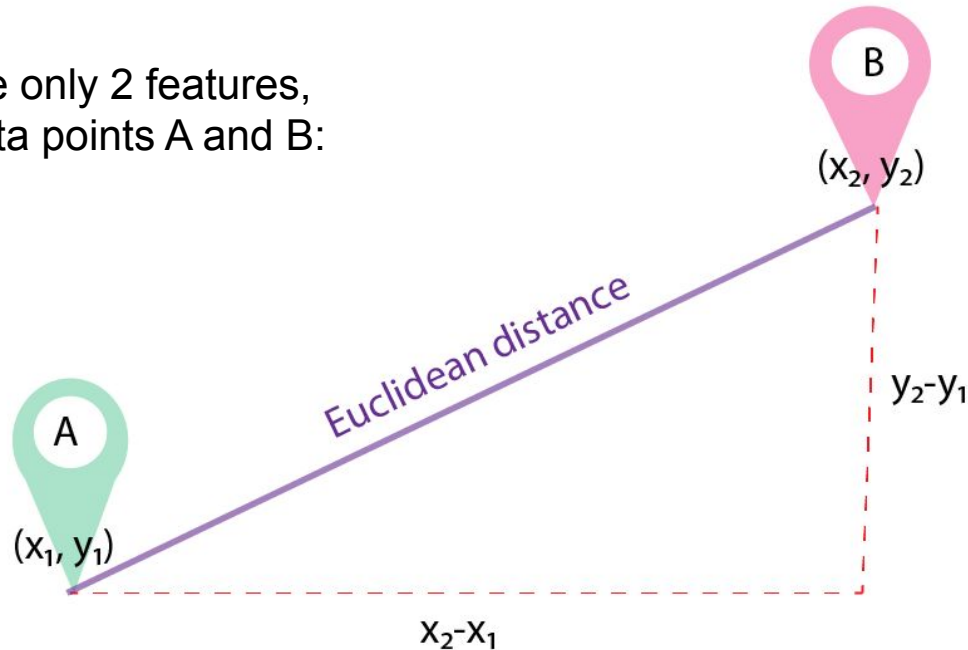
Distance:

- (popular) Euclidean distance: $d(x, x') = \sqrt{\sum_{i=1}^d (x_i - x'_i)^2}$
- Manhattan distance : $d(x, x') = \sum_{i=1}^d |x_i - x'_i|$

A total of d dimensions/features: i from 1 to d

Euclidean distance

Suppose there are only 2 features,
and we have 2 data points A and B:



Q: If we shift a feature, would the distance change?

no

Q: What about scaling a feature?

yes

Issue: features in different scales

Suppose our train set:

Index	Weight (lbs)	Shoe size
0	130	9
1	180	10.5
2	100	6
3	210	12
4	155	7.5
5	170	9.5
6	90	7

Features in different scales can be problematic:
“weight” is the dominance in the distance.

- Features having different scale can be problematic. (e.g., weights in lbs vs shoe size)

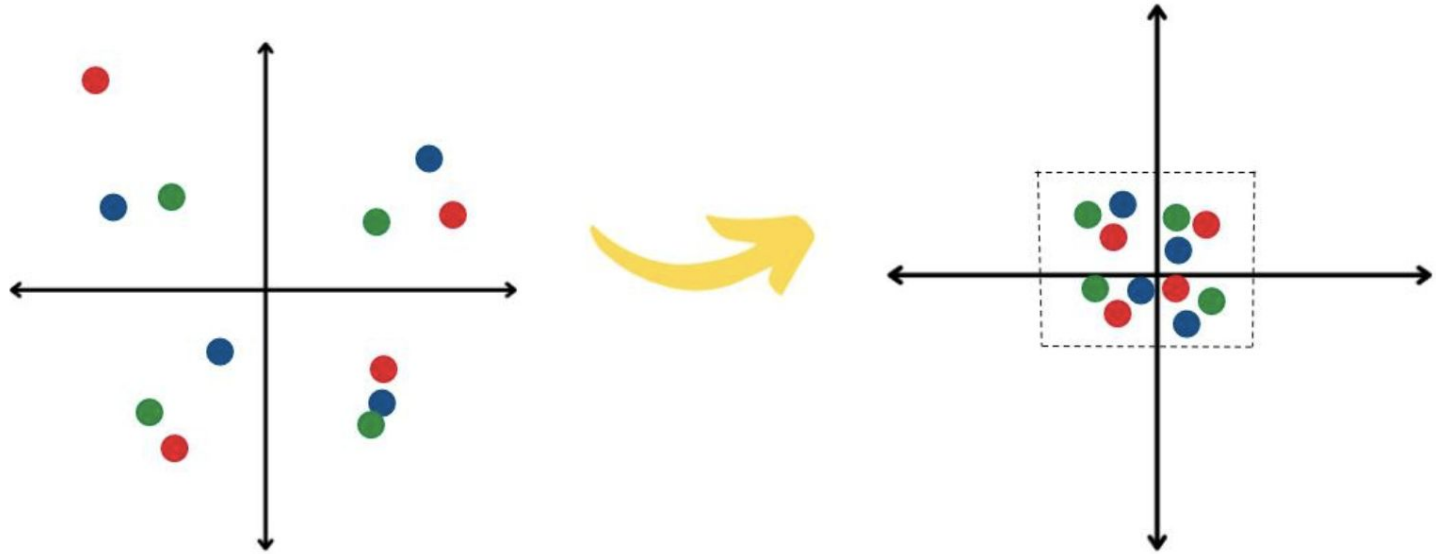
- [Definition] **Standardization**

- For each feature f , compute $\mu_f = \frac{1}{m} \sum_{i=1}^m x_f^{(i)}$, $\sigma_f = \sqrt{\frac{1}{m} \sum_{i=1}^m (x_f^{(i)} - \mu_f)^2}$
- Then, transform the data by $\forall f \in \{1, \dots, d\}, \forall i \in \{1, \dots, m\}, x_f^{(i)} \leftarrow \frac{x_f^{(i)} - \mu_f}{\sigma_f}$

after transformation, each feature has mean 0 and variance 1

- Be sure to keep the “standardize” function and apply it to the test points.
 - Save $\{(\mu_f, \sigma_f)\}_{f=1}^d$
 - For test point x^* , apply $x_f^* \leftarrow \frac{x_f^* - \mu_f}{\sigma_f}, \forall f$

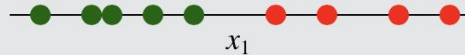
Standardization



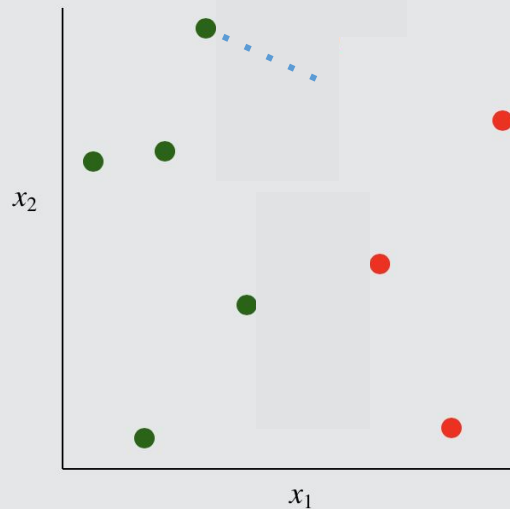
After transformation, each feature has mean 0 and variance 1.

- Given: labeled data D
- Training
 - Compute and save $\{(\mu_f, \sigma_f)\}_{f=1}^d$
 - Compute and save standardization of D
- Test
 - Given x^* , apply standardization $x_f^* \leftarrow \frac{x_f^* - \mu_f}{\sigma_f}, \forall f$
 - Compute k nearest neighbors $N(x^*)$
 - Predict by majority vote label in $N(x^*)$ (average label for regression tasks)

here's a case in which there is one relevant feature x_1 and a 1-NN rule classifies each instance correctly



consider the effect of an irrelevant feature x_2 on distances and nearest neighbors

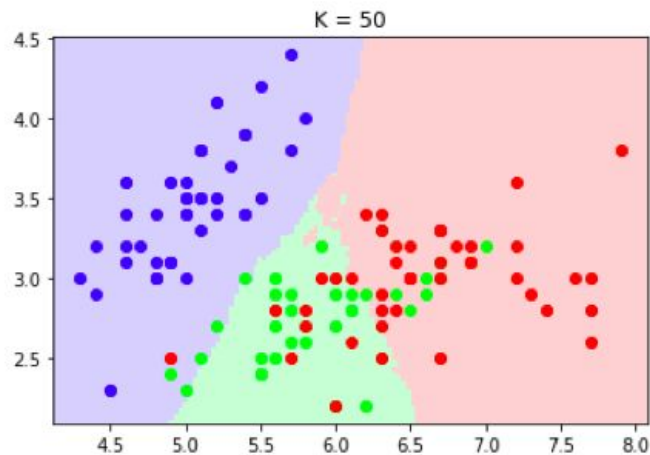
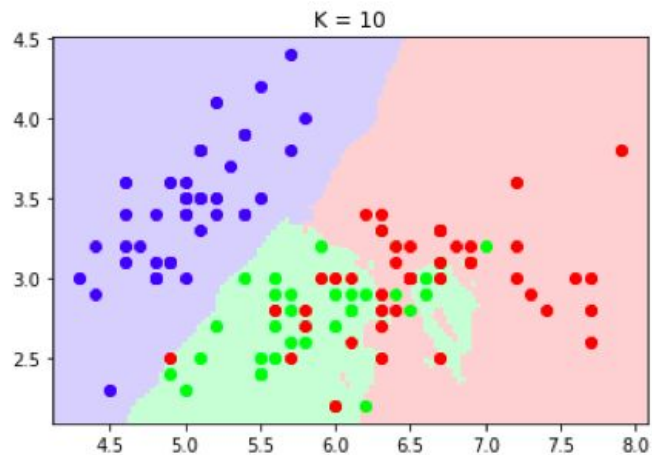
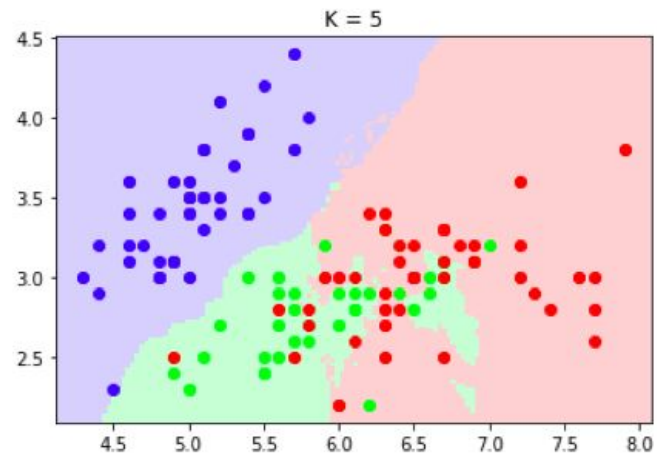
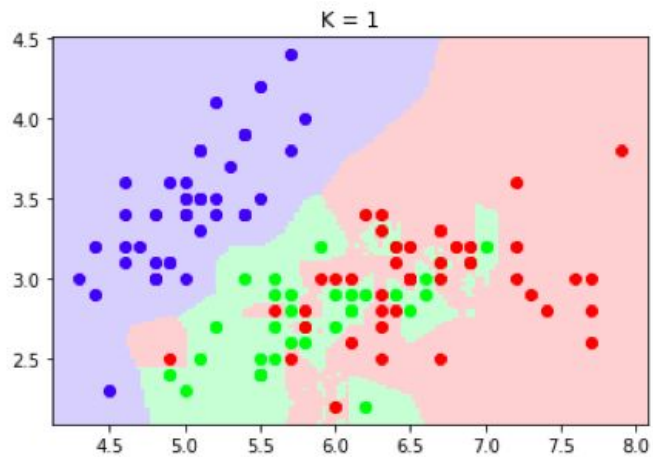


Q: how did we deal with irrelevant features in decision trees?

not all features are used because (a) we stop adding features when having 0 local accuracy, (b) prune the tree

Issue 2: choosing k

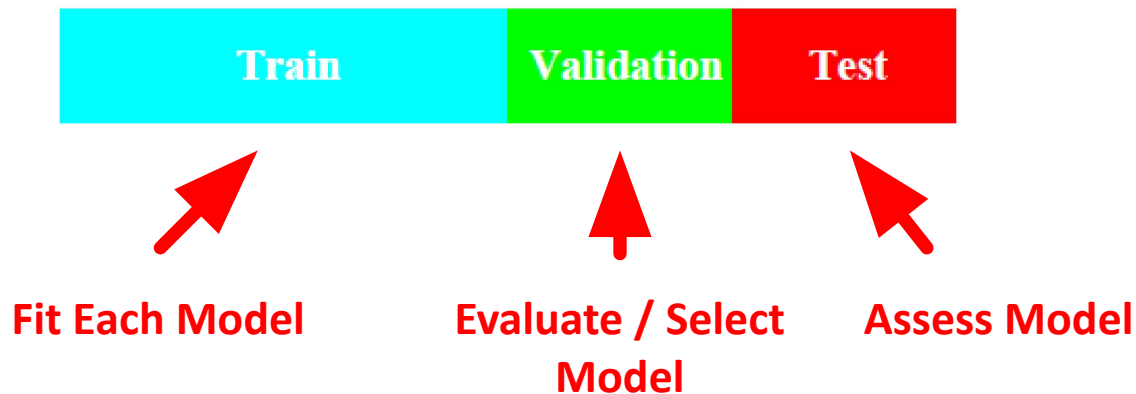
25



	Decision Tree	k-NN
• Interpretability	good	bad
• Sensitivity to irrelevant features	low	high
• train time	$O(dm^2 + dm \log m)$	$O(dm)$
• test time	depth of the tree worst: $O(\min\{d, m\})$ best: $\log(m)$	$O(m(d + \log(m)))$ bad
• test time space complexity	worst: $O(m)$ in general: much smaller	$\Theta(dm)$

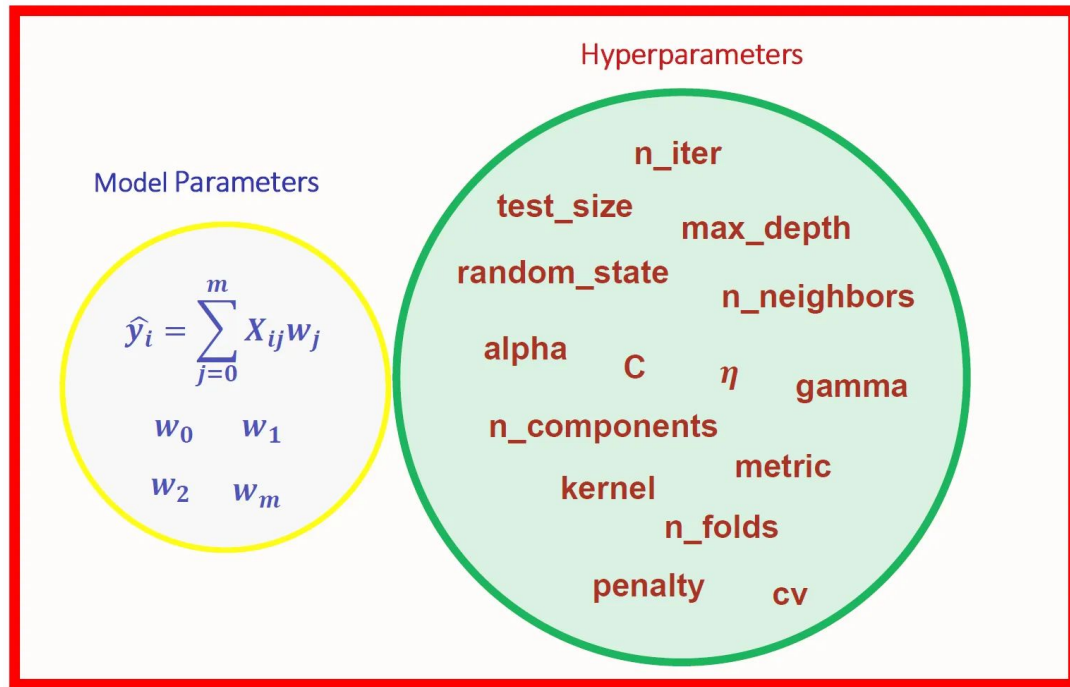
Model Selection and Evaluation

Partition your data into Train-Validation-Test sets



- Ideally, Test set is kept in a “vault” and **only peek at it once model is selected**
- Small dataset: 50% Training, 25% Validation, 25% Test (very loose rule set by statisticians)
- For large data (say a few thousands), 80-10-10 is usually fine.

Hyperparameters vs Parameters



hyperparameter: parameters of the model that are not trained automatically by ML algorithms (e.g., k in KNN).

parameters: those that are trained automatically (e.g., tree structures in decision tree)

Validation set method:

- For each hyperparameter $h \in H$
 - Train \hat{f} on train set with h
 - Compute the error rate of \hat{f} on validation set
- Choose the best performing hyperparameter h^*
- Use h^* to retrain the final model \hat{f}^* with both train and validation set.
- Finally, evaluate \hat{f}^* on test set to estimate its future performance.

Pro tip

- Do not use arithmetic grids; use geometric grids.

Don't
Do

$k = 1, 3, 5, 7, 9, \dots$

$k = 1, 2, 4, 8, 16, \dots$

K-fold cross validation

- Randomly partition train set S into K disjoint sets; call them $\text{fold}_1, \dots, \text{fold}_K$
- For each hyperparameter $h \in \{1, \dots, H\}$
 - For each $k \in \{1, \dots, K\}$
 - train \hat{f}_k^h with $S \setminus \text{fold}_k$
 - measure error rate $e_{h,k}$ of \hat{f}_k^h on fold_k
 - Compute the average error of the above: $\widehat{err}^h = \frac{1}{K} \sum_{k=1}^K e_{h,k}$
- Choose $\hat{h} = \arg \min_h \widehat{err}^h$
- Train \hat{f}^* using S (all the training points) with hyperparameter h
- Finally, evaluate \hat{f}^* on test set to estimate its future performance.

Use when (1) the dataset is small (2) ML algorithm's retraining time complexity is low (e.g., kNN)

5-fold cross validation



Python library for machine learning. Install using Anaconda:



```
$ conda install -c conda-forge scikit-learn
```

Or using PyPi:

```
$ pip install -U scikit-learn
```

Oftentimes, categorical labels come as strings, which aren't easily modeled (e.g., with Naïve Bayes),

```
>>> le = preprocessing.LabelEncoder()
>>> le.fit(["paris", "paris", "tokyo", "amsterdam"])
LabelEncoder()
>>> list(le.classes_)
['amsterdam', 'paris', 'tokyo']
>>> le.transform(["tokyo", "tokyo", "paris"])
array([2, 2, 1]...)
>>> list(le.inverse_transform([2, 2, 1]))
['tokyo', 'tokyo', 'paris']
```

`LabelEncoder` transforms these into integer values, e.g. for categorical distributions

`fit()` is doing the heavy work: create the mapping from string to integers

Can *undo* using `inverse_transform` so we don't have to store two copies of the data

Typical ML workflow starts with *pre-processing* or *transforming* data into some useful form, which Scikit-Learn calls *transformers*:

```
>>> from sklearn.preprocessing import StandardScaler
>>> data = [[0, 0], [0, 0], [1, 1], [1, 1]]
>>> scaler = StandardScaler()
>>> print(scaler.fit(data))
StandardScaler()
>>> print(scaler.mean_)
[0.5 0.5]
>>> print(scaler.transform(data))
[[-1. -1.]
 [-1. -1.]
 [ 1.  1.]
 [ 1.  1.]
```

Standardization:

$$z = \frac{x - \mu}{\sigma}$$

with mean:

$$\mu = \frac{1}{N} \sum_{i=1}^N (x_i)$$

and standard deviation:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

- Features are standardized independently (columns of X)

Easily do cross validation for model selection / evaluation...

```
>>> from sklearn import datasets, linear_model
>>> from sklearn.model_selection import cross_val_score
>>> diabetes = datasets.load_diabetes()
>>> X = diabetes.data[:150]
>>> y = diabetes.target[:150]
>>> lasso = linear_model.Lasso()  Linear Model trained with Lasso prior as regularizer
>>> print(cross_val_score(lasso, X, y, cv=3))
[0.3315057  0.08022103 0.03531816]
```



Models can be fit using the `fit()` function.
E.g., Random Forest Classifier,

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> clf = RandomForestClassifier(random_state=0)
>>> X = [[ 1,  2,  3], # 2 samples, 3 features
...      [11, 12, 13]]
>>> y = [0, 1] # classes of each sample
>>> clf.fit(X, y)
RandomForestClassifier(random_state=0)
```

`fit()` Generally accepts 2 inputs

- Sample matrix X —typically 2d array ($n_{\text{samples}}, n_{\text{features}}$)
- Target values Y —real numbers for regression, integer for classification

Train / evaluate the KNN classifier for each value K,

```
from sklearn.neighbors import KNeighborsClassifier

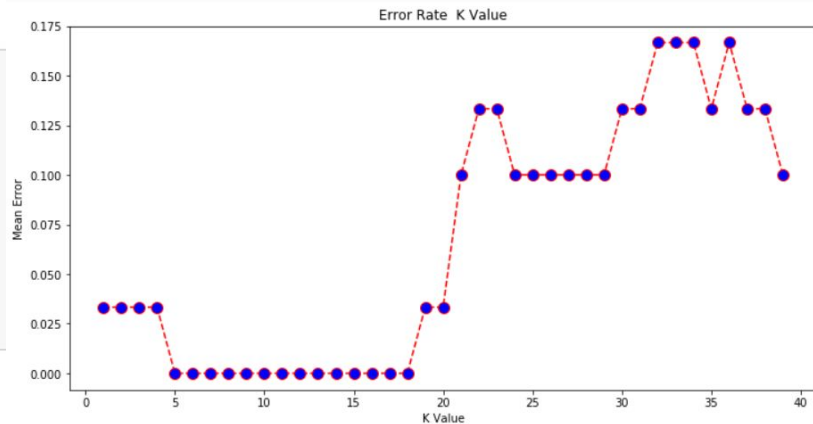
error = []

# Calculating error for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_val)
    error.append(np.mean(pred_i != y_val))
```

↑ vector
operation!

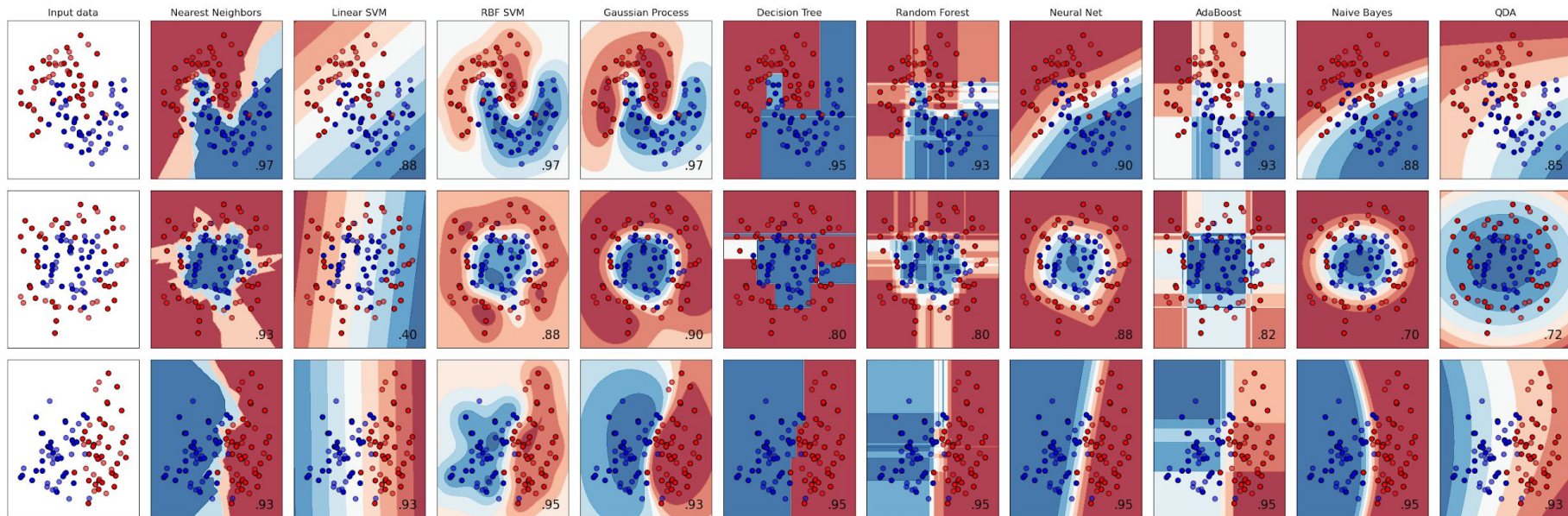
Print error:

```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
        markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```



in practice: use geometric grid like 1,2,4,8,...

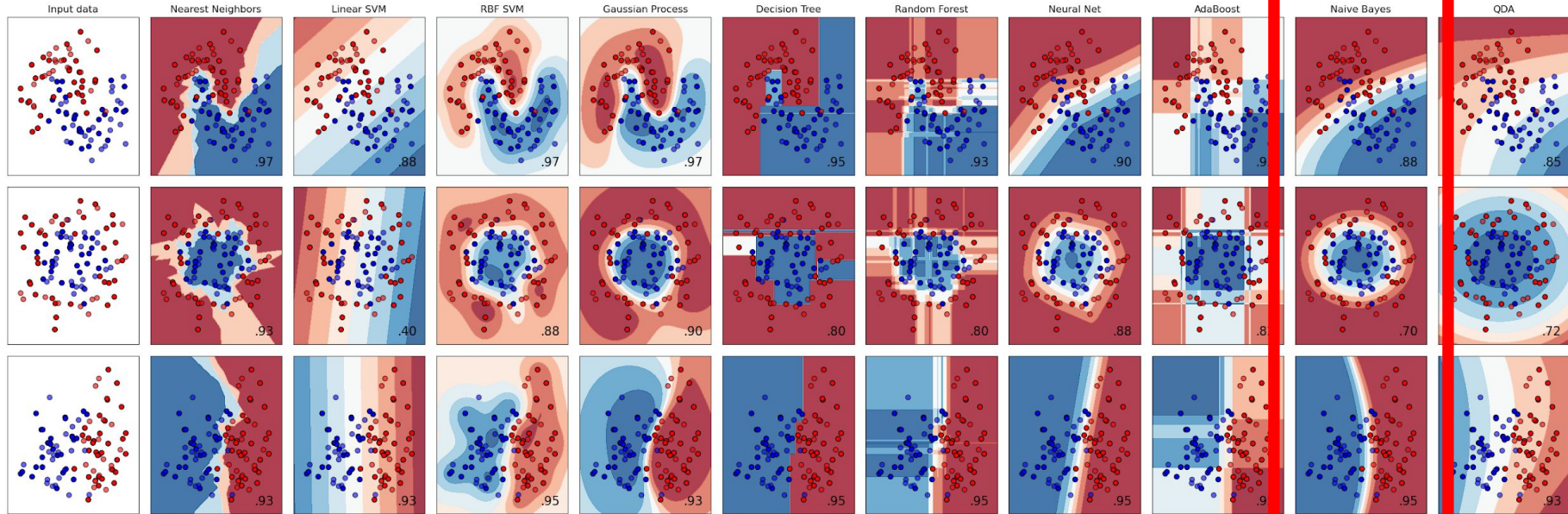
Easily try out *all* the classifiers...



See full code.

Easily try out *all* the classifiers...

Naïve Bayes



See full code.