Introduction to Social Media Analytics (Lec 3)

Hao PENG
Department of Data Science
City University of Hong Kong
https://haoopeng.github.io/

Statistical machine learning

- Machine Learning constructs **algorithms** that can learn from historical data, which is especially useful for **predictions.**
- Statistical Learning is *a branch of Statistics* developed in response to Machine learning; It emphasizes building statistical models for drawing **inferences & correlations** between variables and assessing uncertainty.
- Data Science is the extraction of knowledge from data, using tools from statistics, computer science, engineering . . .

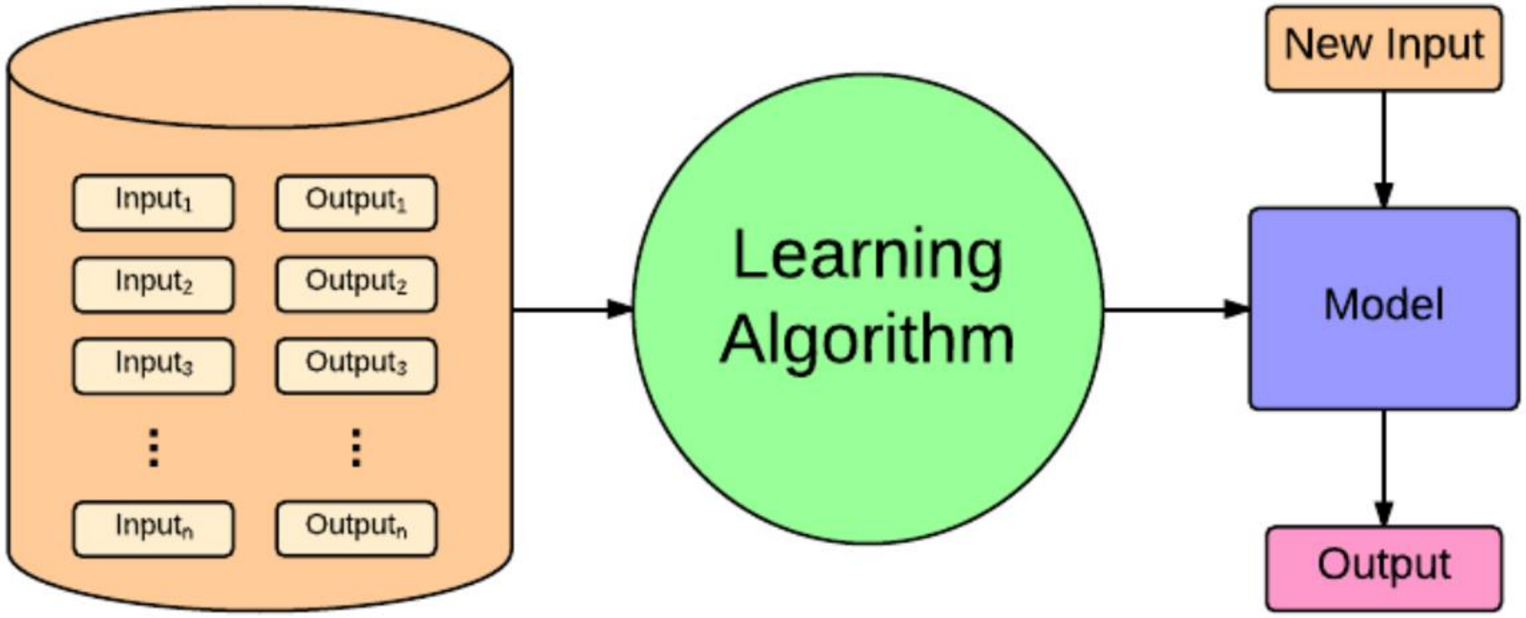
  (there is a "data science" in almost every domain now)

Supervised vs. Unsupervised learning

- **Supervised**: Both inputs (features, a.k.a. covariates, a.k.a. independent variables) and outputs (labels, a.k.a. response, a.k.a. dependent variable) are in training data.

- **Unsupervised**: Only inputs in training data; no outputs.

Supervised learning paradigm



Supervised learning examples

- Use **classification techniques** to identify which accounts are more likely to upgrade to premium services (this helps the salesforce to know which accounts to focus on to sell more).

- Use **regression models** to improve how sales attract prospective clients and uncover what features of the company's services they should highlight in marketing.

- Regression and classification models are examples of **Supervised Learning** techniques.

The Netflix challenge

Netflix offers \$1M USD cash prize to seek for an improved recommender algorithm in October 2006.

- **Training data**:
  - 100M ratings of 17,770 movies by 480K users
  - 6 years of data: 2000-2005
- **Test data**:
  - Last few movie ratings of each user (2.8M)
  - Evaluation via root mean squared error (RMSE)
  - Netflix Cinematch RMSE: 0.9514 (baseline)
- **Competition**:
  - \$1M grand prize for 10% improvement
  - If 10% not met, \$50K annual “Progress Prize” for best improvement

An exercise

Are the following problems supervised or unsupervised?

If they are supervised problems, are they regression (output is numerical) or classification (output is categorical) problems?

- Problem 1: You have a large inventory of identical items. You want to predict how many of these items will sell over the next 3 months.
- Problem 2: You'd like software to examine individual customer accounts, and for each account decide if it has been hacked/compromised.
- Problem 3: Given a database of customer data, automatically discover market segments and group customers into different market segments.

Supervised learning

- Outcome measurement Y
  - output, response, target, dependent variable
- Vector of p predictors / measurements X
  - inputs, regressors, covariates, features, independent variables
- In a regression problem, Y is quantitative
  - E.g., likes, comments, views, age, price
- In a classification problem, Y takes values in a set of labels
  - E.g., gender, education level, personality type
  - We have training data $(x_1, y_1), \ldots, (x_N, y_N)$.
- These are called observations or samples.

Linear regression

How to predict a child's height using parent's height?

- A simple model: $$\hat{y} = \underbrace{\hat{\beta}_0}_{\text{intercept}} + \underbrace{\hat{\beta}_1}_{\text{slope}} x$$
- Find the **best** fit **line** of this form. (what does "best" mean?)

- Why do people believe in a simple prediction model of this kind?
  - Simplicity often means better interpretability
  - OK performance on many tough problems is good
- Before we talk about fitting the linear regression model**,** let's figure out **what is a model and why do we need one**?

Most models are wrong, some are useful when they can provide a good abstraction of how the real world works.

Fitting the regression model

- **Data**: Have n pairs (x_i, y_i), where x_i is the height of parent i and y_i is height of child i

- **Predictions**: $\hat{y}_i$ is our model's prediction of the height of child i

- **Loss function**: First, define a way to measure *error* or *residual* $e_i = y_i - \hat{y}_i$

$$Loss(y, \hat{y}) = (y - \hat{y})^2$$

Fitting the regression model

- **Loss on data**: Estimate parameters β_0 and β_1 by solving least squares problem (suppose the minimizers are $\hat{\beta}_0$ and $\hat{\beta}_1$)

$$\underset{\beta_0,\beta_1}{\text{minimize}} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \left(= \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2 \right)$$

- $e_i = y_i - \widehat{y_i}$ represents the i-th *residual*
- We define the *residual sum of squares* (RSS) as

$$\begin{aligned} RSS &= \textstyle\sum_{i=1}^{n} (y_i - \widehat{y_i})^2 \\ &= (y_1 - \hat{\beta}_0 - \hat{\beta}_1 x_1)^2 + \cdots + (y_n - \hat{\beta}_0 - \hat{\beta}_1 x_n)^2 \end{aligned}$$

How to find solution in loss function?

- We can use the Python LinearRegression as a blackbox to find the $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize the RSS.
- People in the pre-computer age use an exact formula:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \text{ and } \hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$$

remark: every least squares regression line passes ($\bar{x}$, $\bar{y}$)

- Modern packages use optimization techniques.

Multivariate linear regression

In addition to scalar input $x \in \mathbb{R}$, we can consider vector input $x \in \mathbb{R}^p$

- p is feature dimension of input (sometimes use d for *dimension*)
- Inputs of form

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}$$

- Call x the *covariates*, *independent variables*, *explanatory variables*, *features*, *attributes* or *predictor variables*
  - Note that variables are usually NOT independent of one another

Fitting a multivariate linear regression

- Making predictions using

$$\hat{y} = \hat{\beta}_0 + \sum_{j=1}^{p} \hat{\beta}_j x_j$$

- Fit as in single variable case. Solve (least squares criterion or OLS)

$$\underset{\beta_0 \in \mathbb{R}, \beta \in \mathbb{R}^p}{\text{minimize}} \sum_{i=1}^{n} (y_i - \beta_0 - \beta^T x_i)^2$$

- Since each observation has p values, $x_i = (x_{i1}, \cdots, x_{ip})^T$, where $i = 1, \cdots, n$. In matrix notation, let $y = (y_1 \cdots, y_n)^T$ and

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & x_{23} & \dots & x_{2p} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & x_{n3} & \dots & x_{np} \end{bmatrix}$$

where x_{ij} is the ith observation of the jth variable

Assessing model performance

- *Residual sum of squares* (RSS), also known as the sum of squared residuals (SSR) or the sum of squared errors (SSE):

$$RSS = \sum_{i=1}^{n} (y_i - \widehat{y_i})^2$$

- *Root mean square error* (RMSE) and MSE*:*

MSE = RSS / n

RMSE = sqrt (MSE)

- *Residual standard error* (RSE):

$$RSE = \sqrt{\frac{1}{n-p-1} RSS}$$

p = # of features (p = 1 in simple linear regression)

n – p – 1 = the residual degrees of freedom

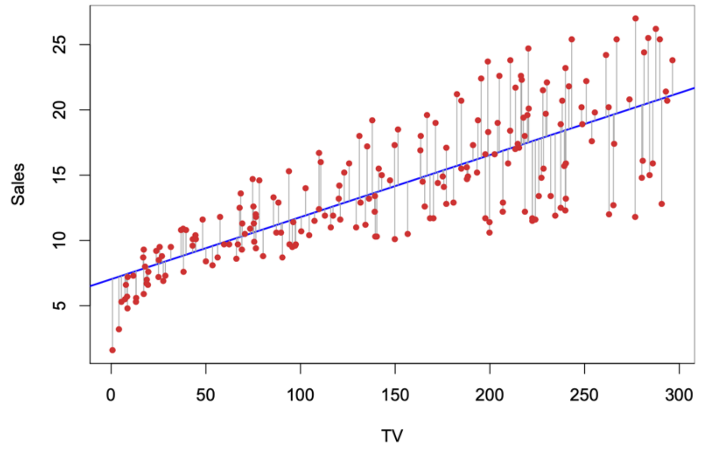
https://www.rdocumentation.org/packages/sjstats/versions/0.17.2/topics/cv

Assessing model performance

- RMSE has the nice property of being in the same unit as the response variable. Lower values of RMSE indicate better fit.

- RMSE can be interpreted as the standard deviation of the unexplained variance / residuals.

- RMSE is a good measure of how accurately the model predicts the response. It's one of the most important criteria if the main purpose of the model is prediction.



- RMSE = 3.2
- Unit of sales: thousand USD

- Actual sales in each market deviate from the regression line by approximately 3,200 USD, on average.

Assessing model performance

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS}$$

- TSS measures the total variance in the response Y. It can be thought of as the amount of **variability inherent** in the response before the regression.

- RSS measures the amount of variability that is left **unexplained after performing the regression**. TSS − RSS measures the amount of variability in the response that is **explained by performing the regression**.

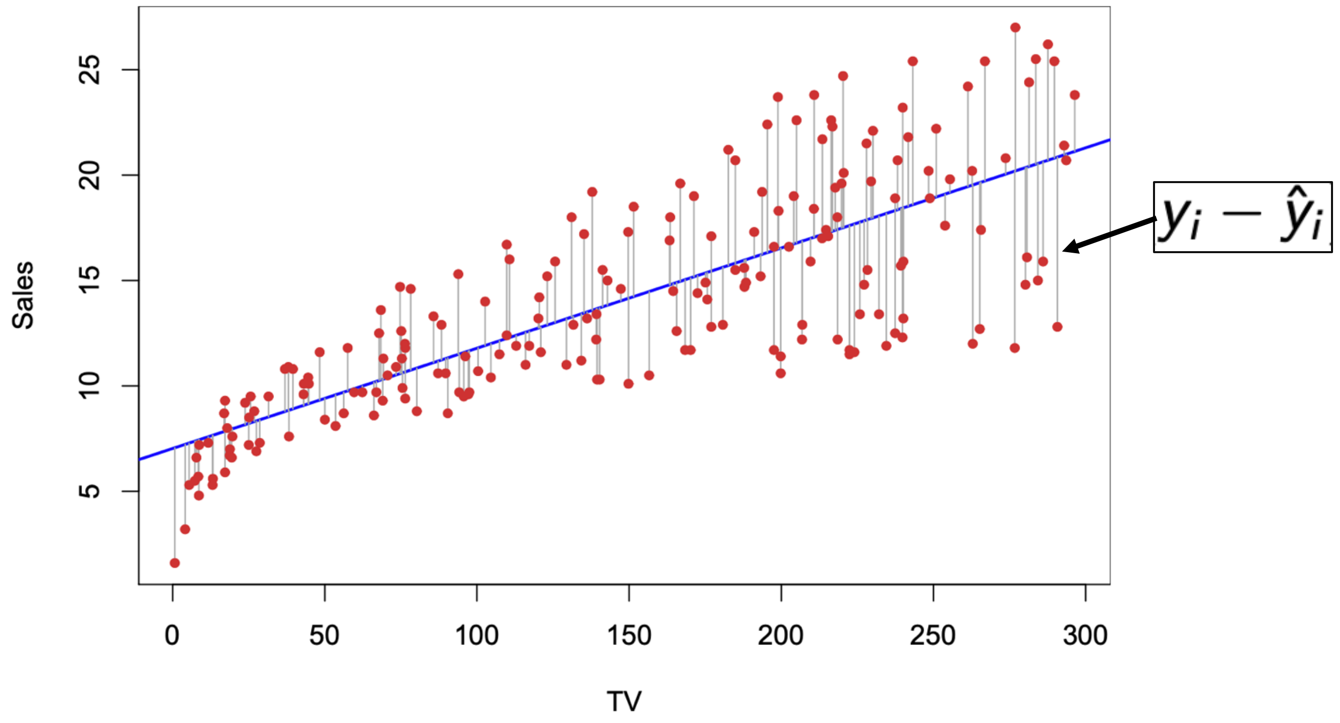
- R^2 measures the *proportion of variability in* Y *that can be explained by* X.

- R^2 is close to 1 indicates that a large proportion of the variability in the response is explained by the regression. R^2 close to 0 often indicates wrong model assumption.

- R^2 is a goodness-of-fit measure; but is not often used to compare different models.

- In different fields, *good* R^2 values are vastly different. E.g., even with a great model, studies that try to predict human behaviors generally have R^2 less than 0.5.
  - Why? People are just harder to predict than things like physical processes.

Example: least square fit

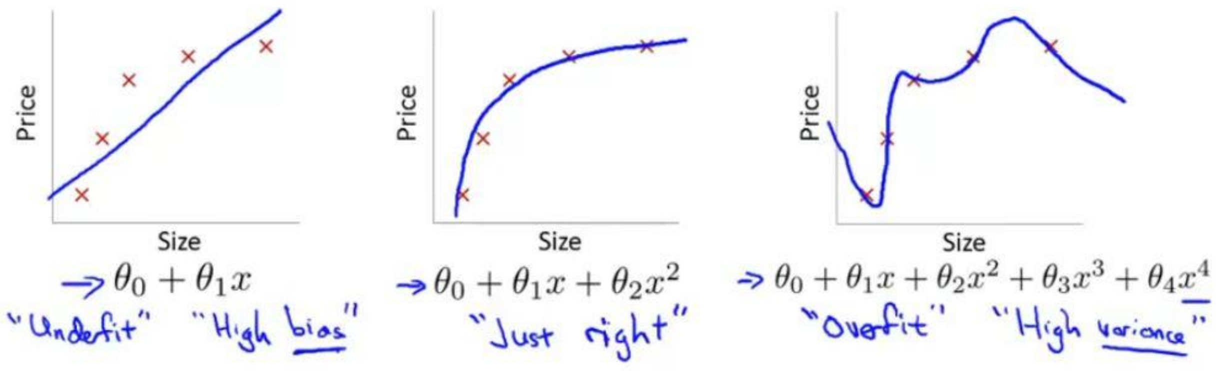


- R^2 = 0.61.
- 61% the variability in sales is explained by a linear regression on TV.
- Not affect by the units.

Caveats of comparing R^2

R^2 always increases when more vars are added to the model.
Why?

Example: Linear regression (housing prices)



Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

https://medium.com/@jennifer.zzz/more-features-than-data-points-in-linear-regression-5bcabba6883e

Caveats of comparing R^2

R^2 always increases when more vars are added to the model. Why?

Therefore, R^2 cannot be used for comparing whether one model is better than another. **Adjusted R-squared** imposes a higher penalty on a new variable and thus is a more reliable indicator of model performance. Other commonly used criteria to judge model performance would be **AIC/BIC**.

Linear regression in Python

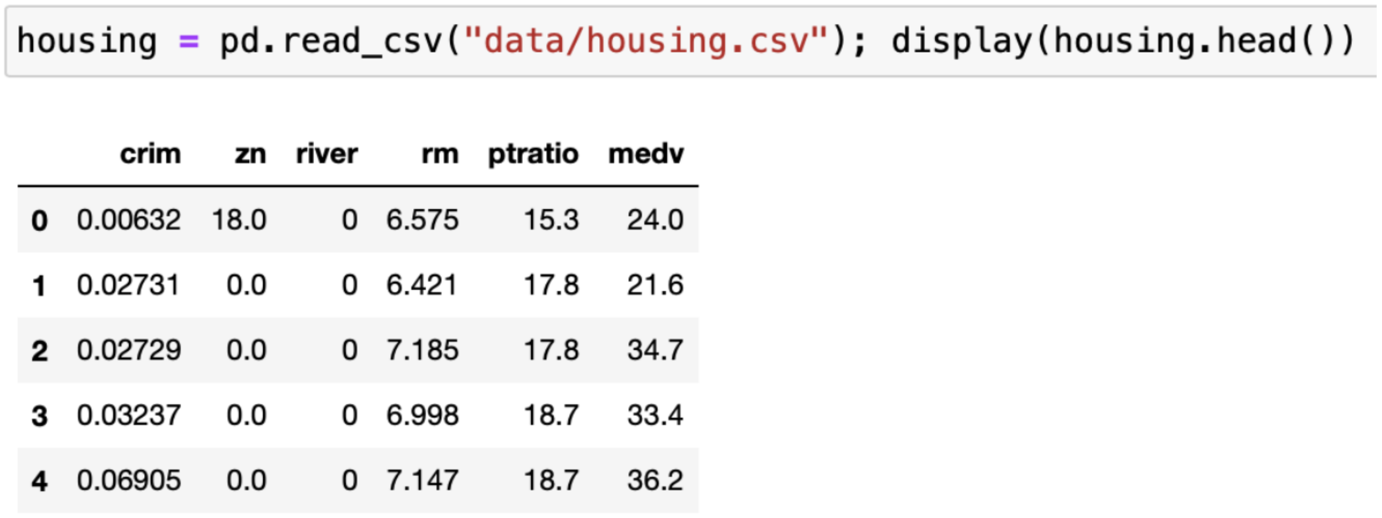
Six basic steps for implementing a linear regression:

1. Import the packages and classes you need.
2. Read in data and do appropriate transformations if necessary.
3. Check relationships between predictors and outcome variable.
4. Create a regression model and fit it with training data.
5. Check the model fitting result to test if the model is satisfactory.
6. Apply the model for predictions on test data.

The package **scikit-learn** is a widely used Python library for machine learning, built on top of NumPy & other packages.

Python example: housing data

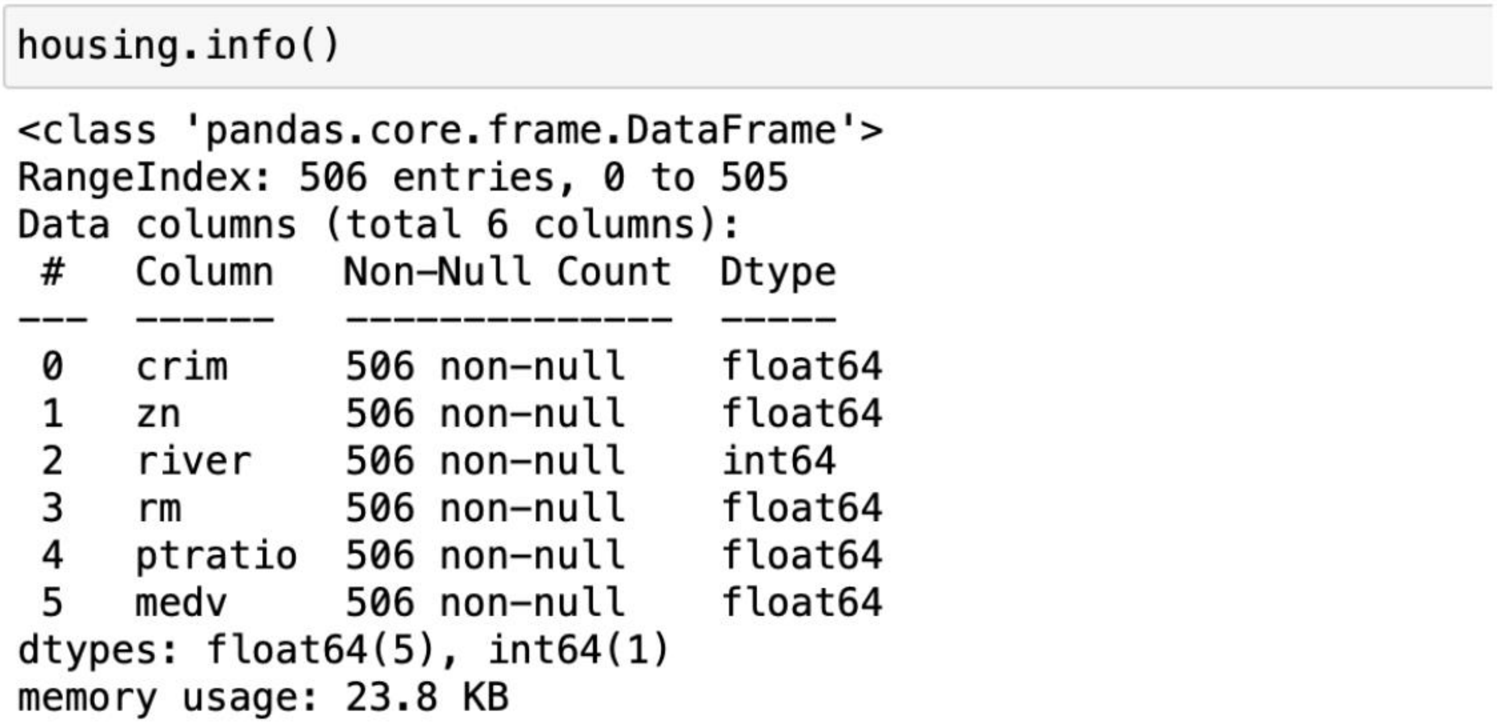
- Output (response, target, dependent) variable is medv, the median home price in different neighborhoods
- covariates include
  - crim: per capita crime rate
  - rm: average number of rooms per dwelling
  - zn: proportion of large lots (zoned for $> 25,000$ feet)
  - river: whether a home is near a river ($x \in \{0, 1\}$)
  - ptratio: pupil-teacher ratio by town



```
housing = pd.read_csv("data/housing.csv"); display(housing.head())
```

	crim	zn	river	rm	ptratio	medv
0	0.00632	18.0	0	6.575	15.3	24.0
1	0.02731	0.0	0	6.421	17.8	21.6
2	0.02729	0.0	0	7.185	17.8	34.7
3	0.03237	0.0	0	6.998	18.7	33.4
4	0.06905	0.0	0	7.147	18.7	36.2

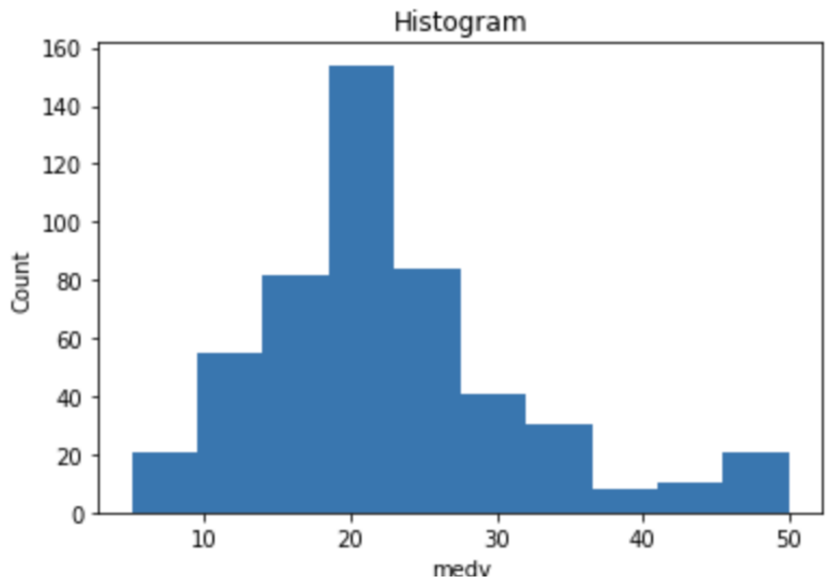
Check the data

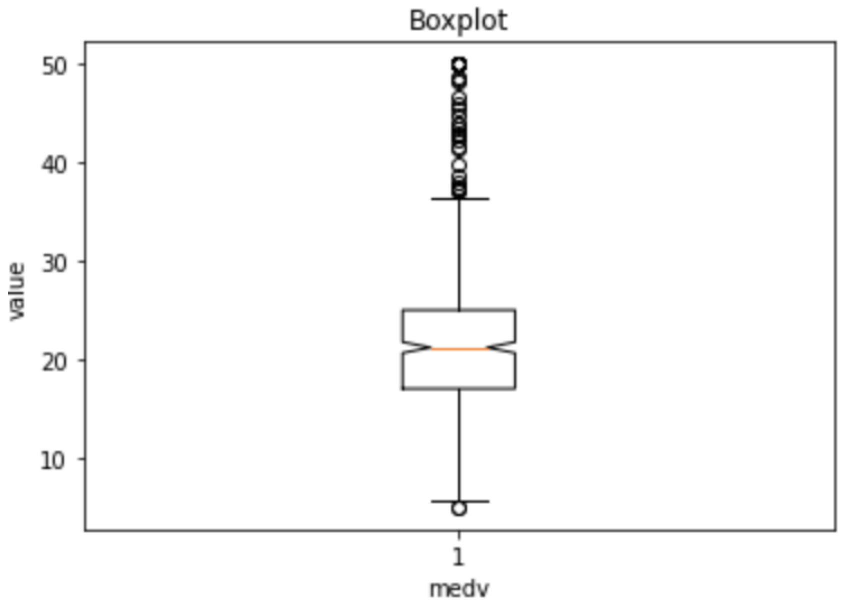


```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   crim     506 non-null    float64
 1   zn       506 non-null    float64
 2   river    506 non-null    int64
 3   rm       506 non-null    float64
 4   ptratio  506 non-null    float64
 5   medv     506 non-null    float64
dtypes: float64(5), int64(1)
memory usage: 23.8 KB
```

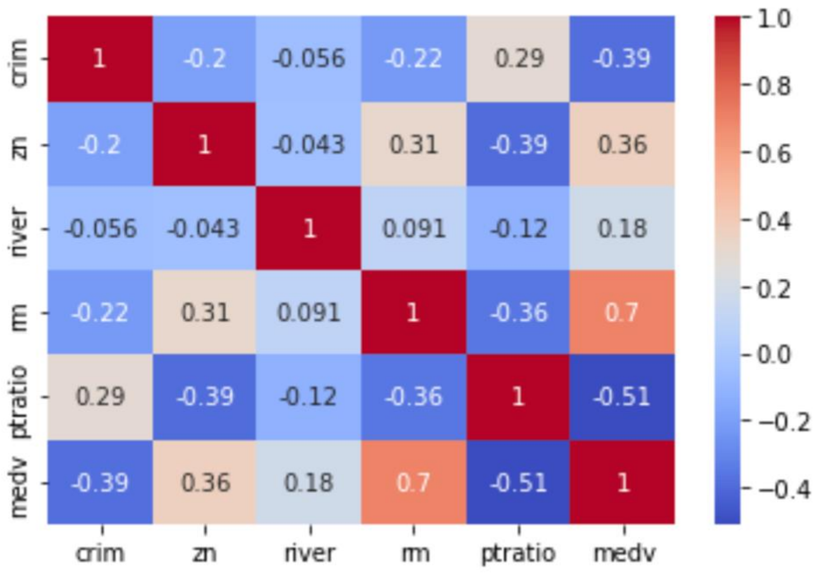
Check the distribution of Y / DV





What can you infer from the plot?

Check the correlation matrix



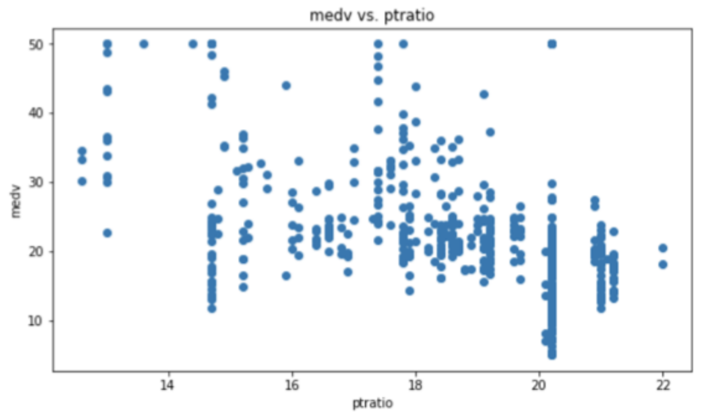
	crim	zn	river	rm	ptratio	medv
crim	1	-0.2	-0.056	-0.22	0.29	-0.39
zn	-0.2	1	-0.043	0.31	-0.39	0.36
river	-0.056	-0.043	1	0.091	-0.12	0.18
rm	-0.22	0.31	0.091	1	-0.36	0.7
ptratio	0.29	-0.39	-0.12	-0.36	1	-0.51
medv	-0.39	0.36	0.18	0.7	-0.51	1

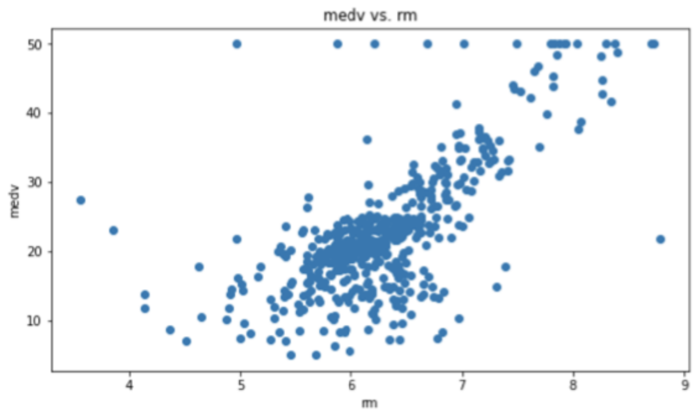
- **medv**: median home price
- **crim**: per capita crime rate
- **rm**: avg. number of rooms per dwelling
- **zn**: proportion of large lots
- **river**: whether a home is near a river
- **ptratio**: pupil-teacher ratio by town

Check independent variables

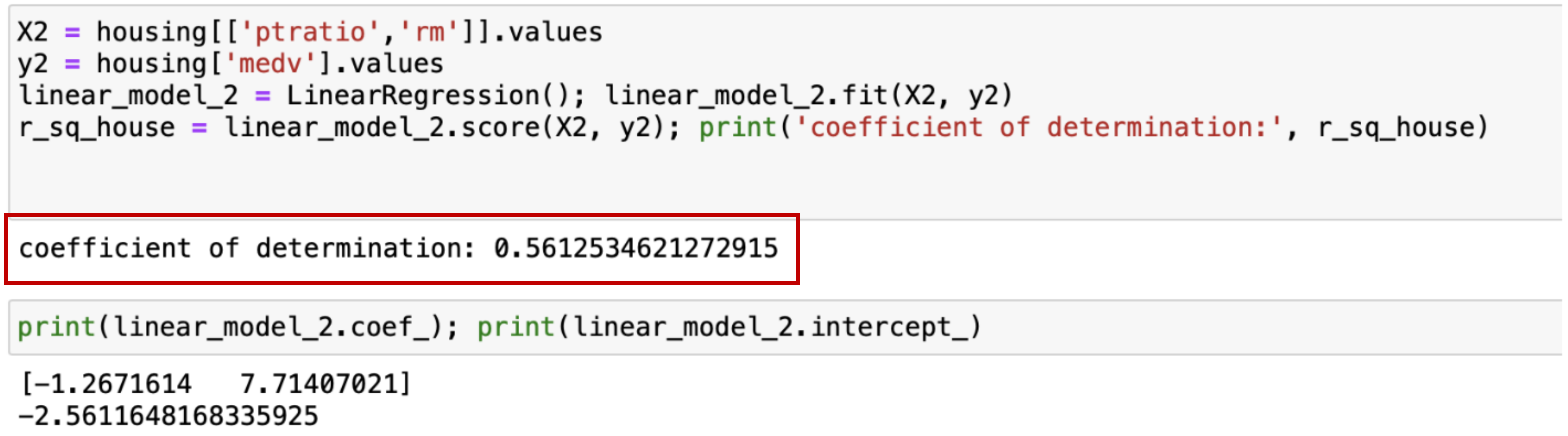
- For illustration purpose, we only use *ptratio* and *rm* as covariates.
- It looks like *ptratio* has negative correlation with *medv* and *rm* has positive correlation with *medv*.
- If for some reason, we have to choose only one of the two predictors, which one would you like to choose?

```
plt.figure(figsize=(20, 5))
features = ['ptratio', 'rm']; target = housing['medv']
for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1) # subplot(nrows, ncols, index). Three separate integers describing
    #the position of the subplot. If the three integers are nrows, ncols, and index in order,
    #the subplot will take the index position on a grid with nrows rows and ncols columns.
    #index starts at 1 in the upper left corner and increases to the right.
    x = housing[col]; y = target
    plt.scatter(x, y)
    plt.title('medv' + " vs. " + col)
    plt.xlabel(col); plt.ylabel('medv')
```





Model fitting using sklearn



```
X2 = housing[['ptratio','rm']].values
y2 = housing['medv'].values
linear_model_2 = LinearRegression(); linear_model_2.fit(X2, y2)
r_sq_house = linear_model_2.score(X2, y2); print('coefficient of determination:', r_sq_house)
```

```
coefficient of determination: 0.5612534621272915
```

```
print(linear_model_2.coef_); print(linear_model_2.intercept_)
```

```
[-1.2671614   7.71407021]
-2.5611648168335925
```

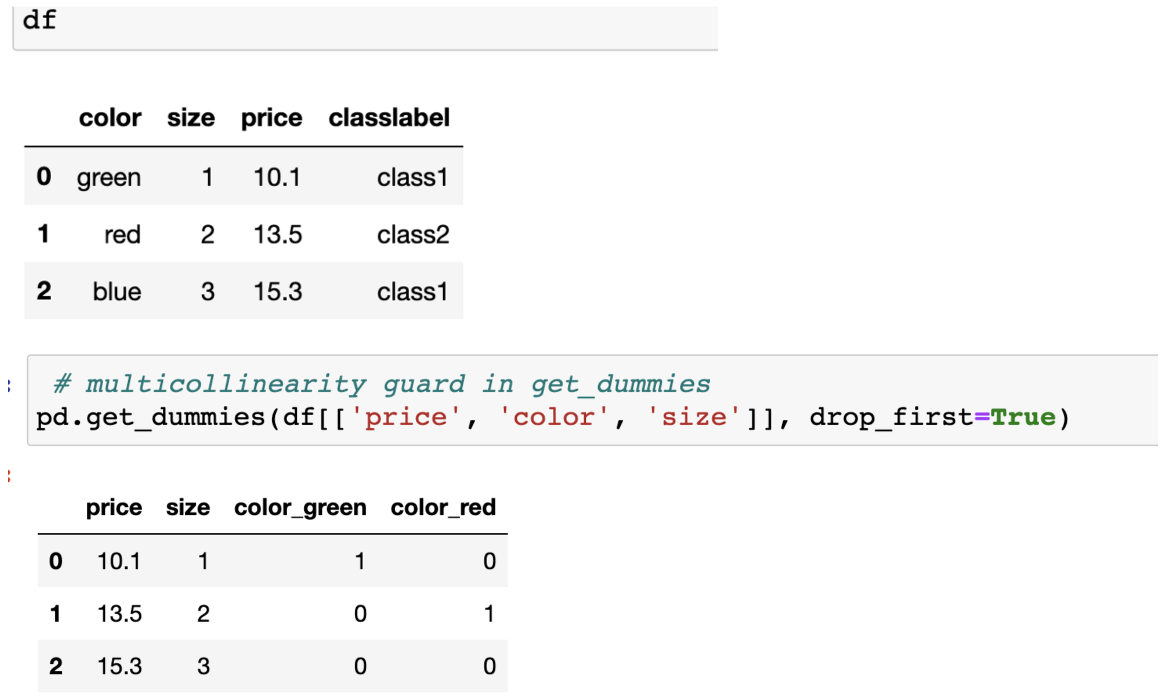
Figure 4: Fitting a multiple linear regression model

- Try regress *medv* on *rm* only, do you see a larger or smaller R square?

Adding categorical variables

Categorical predictors:

- Sometimes, categorical variables can be useful in making predictions.
- We usually code qualitative variables by *dummy variables* (variables taking values 0 and 1). Why are they called dummy variables?
- Suppose you have a categorical variable with three *levels*. We need to code it with two dummy variables.



```
df
```

	color	size	price	classlabel
0	green	1	10.1	class1
1	red	2	13.5	class2
2	blue	3	15.3	class1

```
# multicollinearity guard in get_dummies
pd.get_dummies(df[['price', 'color', 'size']], drop_first=True)
```

	price	size	color_green	color_red
0	10.1	1	1	0
1	13.5	2	0	1
2	15.3	3	0	0

Classification models

- **Classification**: supervised learning when outcomes are categorical (a.k.a. qualitative)
- The categorical outcomes (responses) are usually called *class labels*.
- Both classification and regression are supervised learning
- Classification is perhaps the most widely used machine learning methods. Examples include email spam filter, credit card fraud detection, automatic cancer diagnosis, etc.

What is the usual objective?

- Binary classification is the most common classification scenario
- Features $X \in R^p$ and class labels $Y \in \{0, 1\}$
- A classifier h is some function (usually data-dependent function) that maps the feature space into the label space. One can think of a classifier as a data-dependent partition of the feature space

Why not using linear regression?

- A general remark: before inventing new methods, we should ask why the existing ones do not suffice
- When the outcome variable has more than 2 categories. For example, an `income` variable has three levels: type A, type B, and type C
  - if we code $type\ A = 1$, $type\ B = 2$, $type\ C = 3$, and run linear regression, then
  - i). we have endorsed an ordering in the types
  - ii). we assumed the same difference between pairs
  - an equally reasonable coding $type\ C = 1$, $type\ B = 2$, $type\ A = 3$ will imply a totally different relationship among the three types
  - each of these codings will lead to different predictions
- When the outcome variable has 2 categories
  - we can introduce *dummy variables*
  - and cut the predicted y's at some level, i.e., declare prediction above that level of class 1, and 0 otherwise
  - but this approach is usually inferior to methods that specifically designed for classification

Logistic regression

- Model the conditional probability $P(Y = 1|X = x)$ (compare with linear model)
- The logistic (a.k.a. sigmoid) function

$$f(x) = \frac{e^{\beta_0+\beta_1 x}}{1 + e^{\beta_0+\beta_1 x}}$$

- Logistic regression model: $P(Y = 1|X = x) = f(x)$.

What's the unique characteristic of the sigmoid function f(x)?
Why can we use it to model a binary DV?

How to fit a logistic regression model to data?

Fitting Logistic regression

- The coefficients β_0 and β_1 in the sigmoid function are unknown
- Need to estimate them from training data
- Q: recall linear regression, what criterion did we use to find the coefficient estimates?
- Given training data (pairs are independent of each other)

$$\{(x_1, y_1), \cdots, (x_n, y_n)\}$$

- And let $p(x) = P(Y = 1|X = x)$. We would like to find $\hat{\beta}_0$ and $\hat{\beta}_1$ such that they maximize the *likelihood function* $l(\beta_0, \beta_1)$:

$$l(\beta_0, \beta_1) = \Pi_{i:y_i=1} p(x_i) \Pi_{i':y_{i'}=0}(1 - p(x_{i'}))$$

$$\xleftarrow{\quad} \quad p(x) = \frac{e^{\beta_0+\beta_1 x}}{1 + e^{\beta_0+\beta_1 x}}$$

- This is called a *maximum likelihood approach*

Python implementation and tutorial (must read!):
https://www.datacamp.com/tutorial/understanding-logistic-regression-python

Interpreting coefficients in logit model

- The sigmoid function f takes values between 0 and 1; perfect for modeling probability
- Under the logistic regression model, the *log-odds* or *logit* is linear in the input variable X:

$$\text{log Odds} = \log\left(\frac{P(Y=1|X=x)}{P(Y=0|X=x)}\right) = \log\,[f(x)\,/\,(1-f(x)] = \beta_0 + \beta_1 X$$

- In some books, the above equation is the definition of logistic regression model, or called *logit model*. These two definitions are equivalent
- For logit function: https://en.wikipedia.org/wiki/Logit

- β_1 can be interpreted as the average change in log-odds associated with a one-unit increase in X
- β_1 does NOT correspond to the change in $P(Y=1|X=x)$ associated with 1 one-unit increase in X

Example

Suppose we collect data for a group of students in a statistics class with variables X_1=hours studied, X_2= undergrad GPA, and Y=receive an A. We fit a logistic regression and produce estimated coefficients $\hat{\beta}_0 = -6$, $\hat{\beta}_1 = 0.05$, $\hat{\beta}_2 = 1$. (Take $Y = 1$ to mean `receive an A`)

- Estimate the probability that a student who studies for 40 hours and has an undergrad GPA of 3.5 to get an A in the class.
- How many hours would the student in the previous part need to study to have a 50% chance of getting an A in the class?

Estimate the probability:

$$\begin{aligned} f(x) &= \frac{e^{-6+0.05x_1+x_2}}{1+e^{-6+0.05x_1+x_2}} \\ &= \frac{e^{-6+0.05*40+3.5}}{1+e^{-6+0.05*40+3.5}} \\ &\approx 0.38 \end{aligned}$$

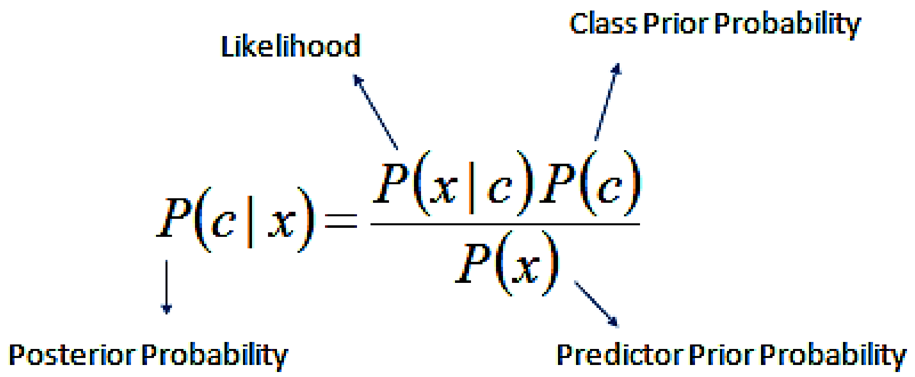
How many hours?

$$\begin{aligned} log(0.5/0.5) &= -6+0.05x_1+x_2 \\ 0 &= -6+0.05x_1+3.5 \\ x_1 &= 50 \end{aligned}$$

A 10 unit increase in X_1 is not associated with 10 x β_1 = 0.5 increase in *P(Y=A)*. But the log (odd ratio) increases by 0.5 [Do the calculation: log (0.38/0.62) = - 0.5].

Naïve Bayes classifier

- The Naïve Bayes classifier is a simple probabilistic classifier which is based on Bayes theorem but with *strong assumptions regarding independence*.
- It is popular in email filtering, spam detection, document categorization.
- Fast, efficient (time and space), scalable, easy to understand.



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

$$P(c|\mathrm{X}) \propto P(x_1|c)\times P(x_2|c)\times \cdots \times P(x_n|c)\times P(c)$$

Prediction: $y = arg\ max\ p(y=C_k)\prod p(x|y=C_k)$

https://towardsdatascience.com/introduction-to-na%C3%AFve-bayes-classifier-fa59e3e24aaf
https://afit-r.github.io/naive_bayes

NB classification example

A dataset of 15 records in the table below are used to train a Naïve Bayes model, and then a prediction is made to a new record *X(B, S).*

Obs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
X1	A	A	A	A	A	B	B	B	B	B	C	C	C	C	C
X2	S	M	M	S	S	S	M	M	L	L	L	M	M	L	L
Y	0	0	1	1	0	0	0	1	1	1	1	1	1	1	0

- First calculate prior and conditional probability:

Prior Probability

P(Y=1) = 9/15   P(Y=0)=6/15

Conditional Probability

P(X1=A|Y=1)=2/9	P(X1=B|Y=1)=3/9	P(X1=C|Y=1)=4/9
P(X2=S|Y=1)=1/9	P(X2=M|Y=1)=4/9	P(X2=L|Y=1)=4/9
P(X1=A|Y=0)=3/6	P(X1=B|Y=0)=2/6	P(X1=C|Y=0)=1/6
P(X2=S|Y=0)=3/6	P(X2=M|Y=0)=2/6	P(X2=L|Y=0)=1/6

If X1 and X2 are individual words and their values represent the number of occurrence in each document, this can be used for text classification. (Y = spam or not, news topics, etc.)

- Prediction:

P(Y=1)P(X1=B|Y=1)P(X2=S|Y=1)=1/45

P(Y=0)P(X1=B|Y=0)P(X2=S|Y=0)=1/15

- P(Y=0)P(X1=B|Y=0)P(X2=S|Y=0)> P(Y=1)P(X1=B|Y=1)P(X2=S|Y=1), so y=0.

K-nearest neighbors (KNN)

- 'KNN': to predict class label for an observation $X = x$, the K training observations that are closest to x are identified. Then x is assigned to the class to which the plurality of these observations belong

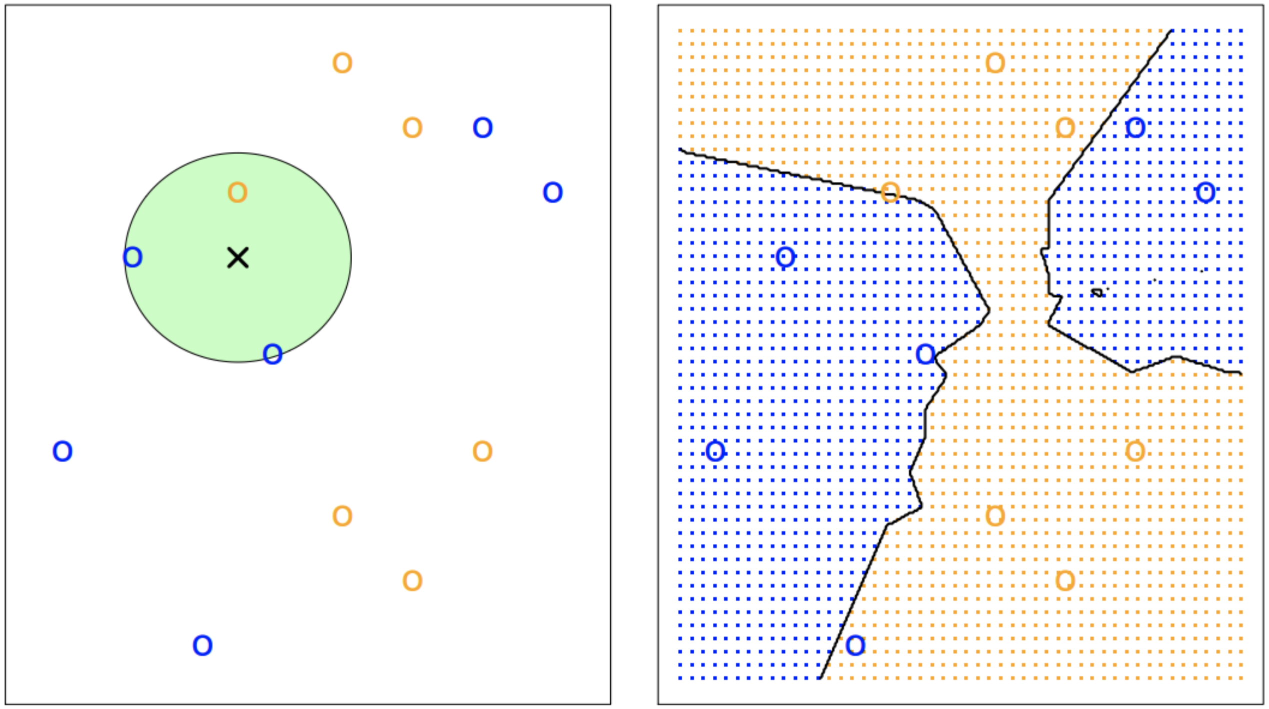
- Can predict the class label for any $x \in R^p$ (including the training observations) in this way
- Special cases: $K = 1$ and $K = n$ (n is the training sample size)

The key is to define a proper distance function.
(Manhattan Distance, Euclidean Distance, Cosine distance, etc.)

https://www.kdnuggets.com/2020/11/most-popular-distance-metrics-knn.html

K-nearest neighbors (KNN)

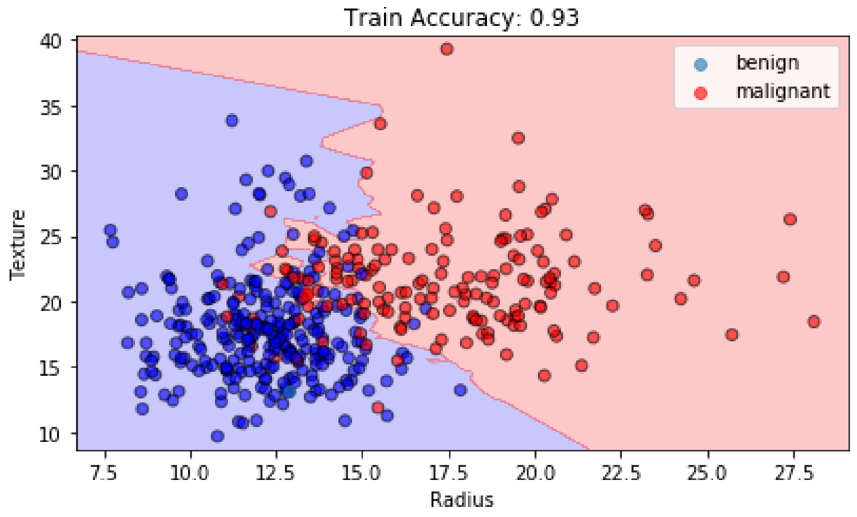
K = 3, Euclidean distance

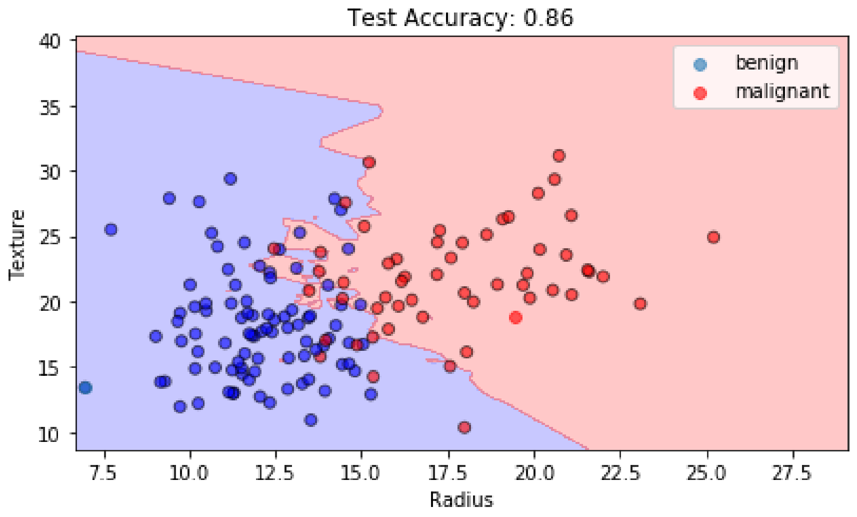


“An Introduction to Statistical Learning” by *Gareth James*, et al. 2013

K-nearest neighbors (KNN)

KNN Classification at K=3



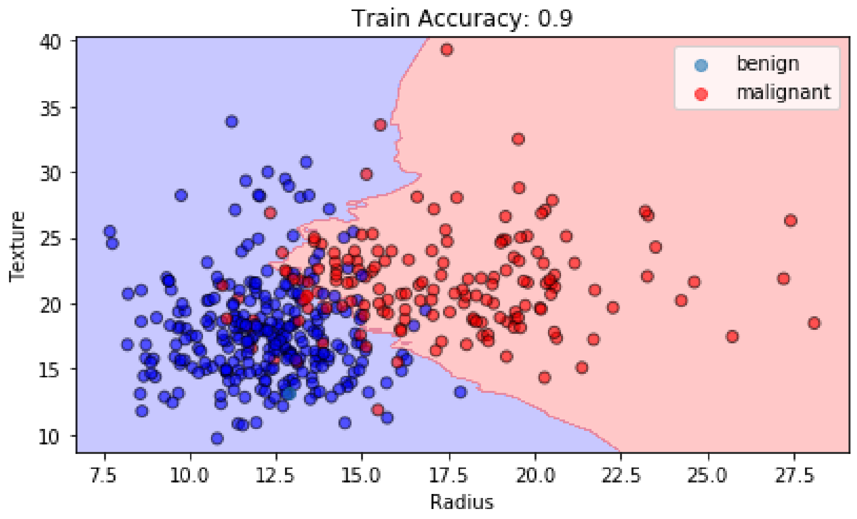


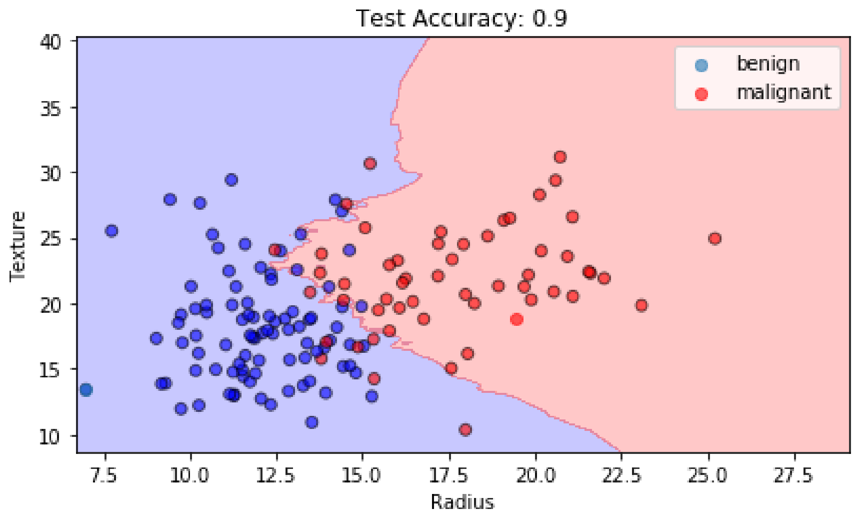
When the value of K is too low, the model picks only the points that are closest to the data sample, thus forming a very complex decision boundary (too rigid). Such a model fails to generalize well on the test data, thereby showing poor results.

https://towardsdatascience.com/k-nearest-neighbors-94395f445221

K-nearest neighbors (KNN)

KNN Classification at K=11



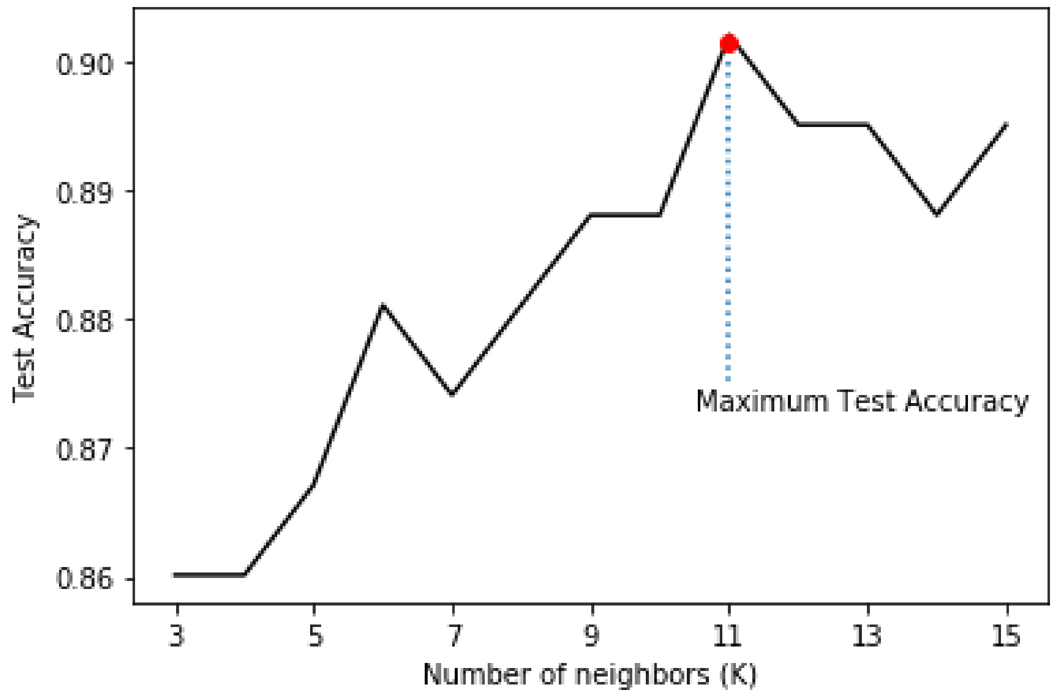


As we increase the number of neighbors, the model starts to generalize better. But increasing K too much may decrease the performance on test data.

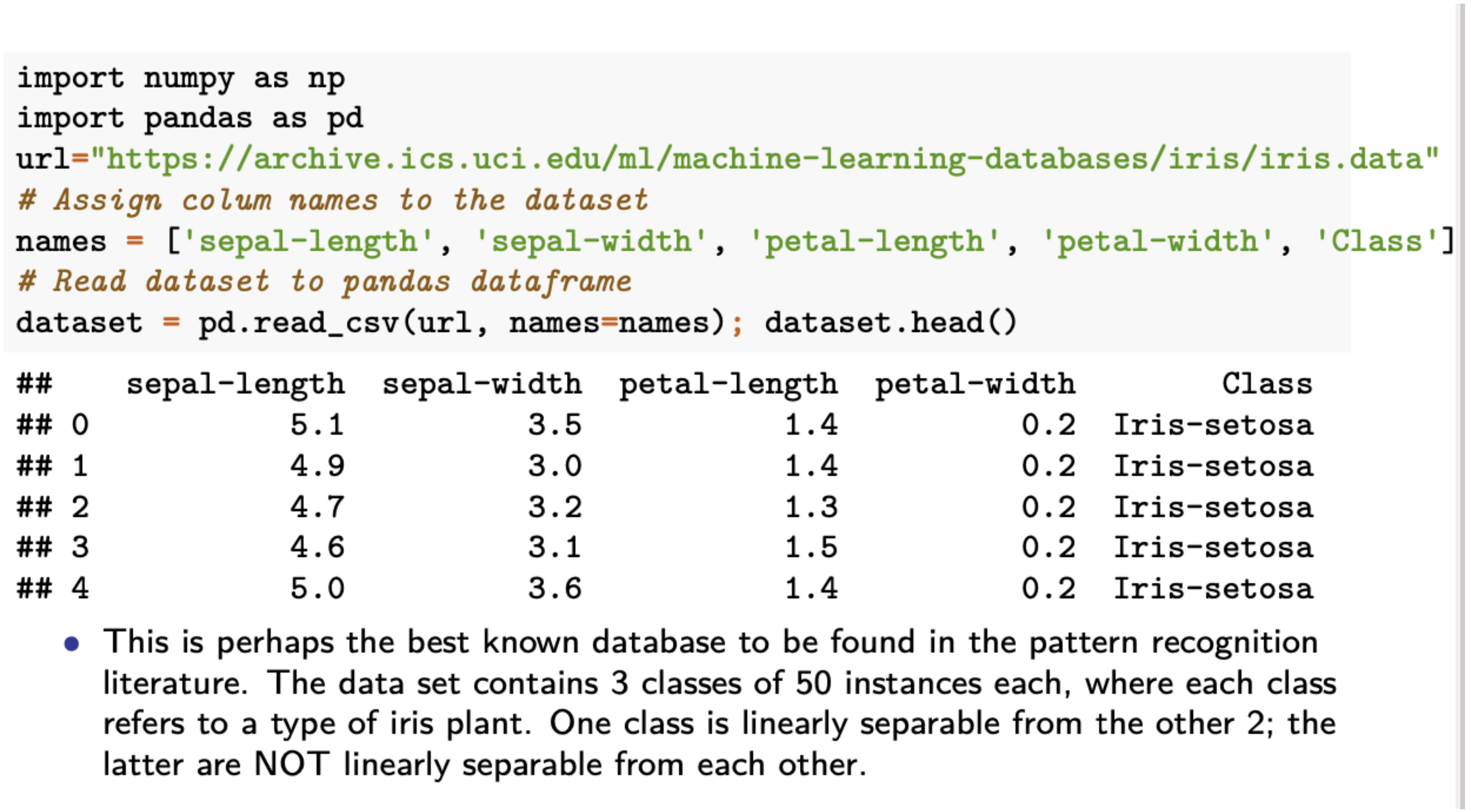
How to pick the best K?

Fine-tuning the parameter K

Train a KNN classifier for each K on the same data:



Python implementation



```python
import numpy as np
import pandas as pd
url="https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
# Assign colum names to the dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
# Read dataset to pandas dataframe
dataset = pd.read_csv(url, names=names); dataset.head()
```

```
##    sepal-length  sepal-width  petal-length  petal-width        Class
## 0           5.1          3.5           1.4          0.2  Iris-setosa
## 1           4.9          3.0           1.4          0.2  Iris-setosa
## 2           4.7          3.2           1.3          0.2  Iris-setosa
## 3           4.6          3.1           1.5          0.2  Iris-setosa
## 4           5.0          3.6           1.4          0.2  Iris-setosa
```

- This is perhaps the best known database to be found in the pattern recognition literature. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Python implementation

```
np.unique(dataset["Class"])
```

```
## array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

```
dataset.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 150 entries, 0 to 149
## Data columns (total 5 columns):
##  #   Column        Non-Null Count  Dtype
## ---  ------        --------------  -----
##  0   sepal-length  150 non-null    float64
##  1   sepal-width   150 non-null    float64
##  2   petal-length  150 non-null    float64
##  3   petal-width   150 non-null    float64
##  4   Class         150 non-null    object
## dtypes: float64(4), object(1)
## memory usage: 6.0+ KB
```

Python implementation

Model training:

```
from sklearn.model_selection import train_test_split
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values
X_train, X_test, y_train, y_test=\
  train_test_split(X, y, test_size=0.20, random_state=5, stratify=y)

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
```

```
## KNeighborsClassifier()
```

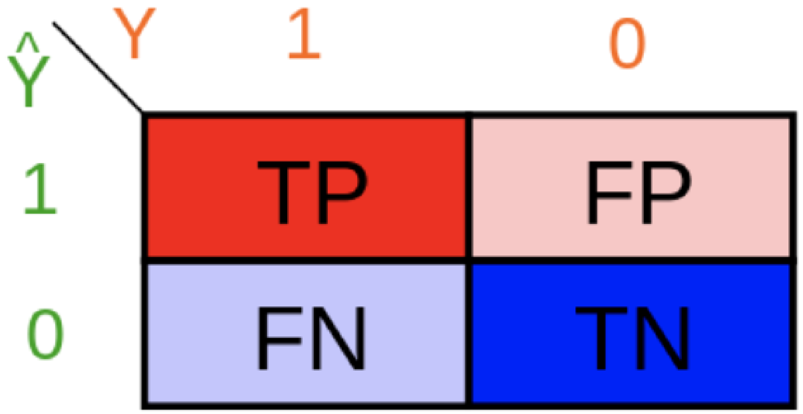
Prediction on test data:

```
y_pred = classifier.predict(X_test)
np.mean(y_pred != y_test)
```

```
## 0.03333333333333333
```

When could accuracy be a bad measure for evaluating model performance?

Confusion matrix



$\hat{Y}$ \ Y	1	0
1	TP	FP
0	FN	TN

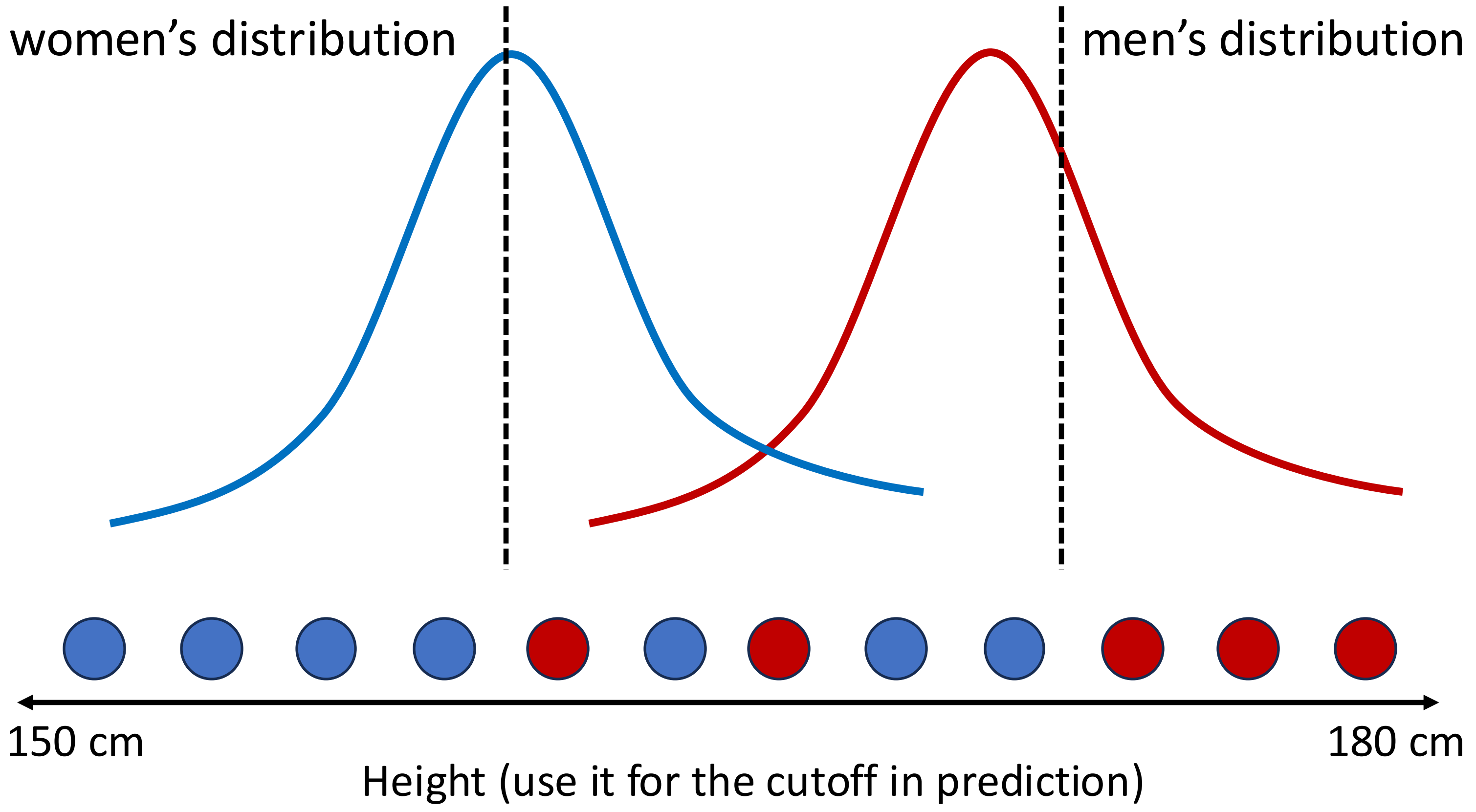
True positive rate:
TP / (TP + FN)
= TP / size of 1

False positive rate:
FP / (FP + TN)
= FP / size of 0

Accuracy = (TP + TN) / n

- Accuracy is misleading in imbalanced datasets.
- For a given threshold, you can calculate a value for each metric.
- When TPR increases, FPR also increases (capturing the trade-off between two types of errors).

Threshold and error trade-offs

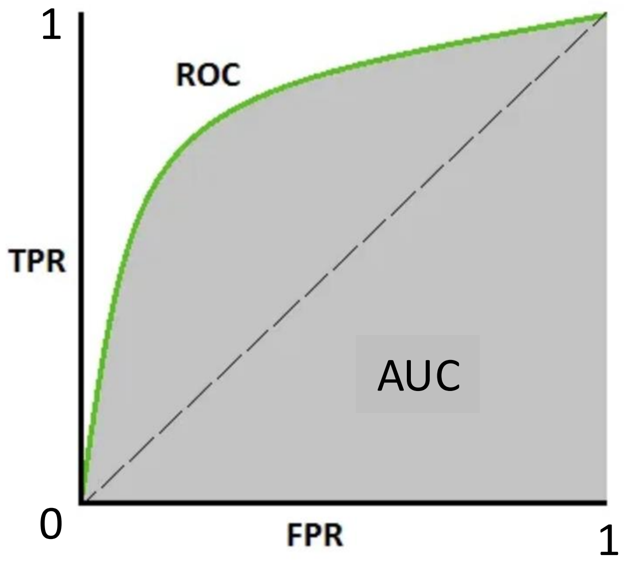


How to pick a height threshold to better classify gender?

Receiver operating characteristic (ROC)

$\hat{Y}$ \ Y	1	0
1	TP	FP
0	FN	TN

$$TPR = \frac{TP}{TP + FN}$$

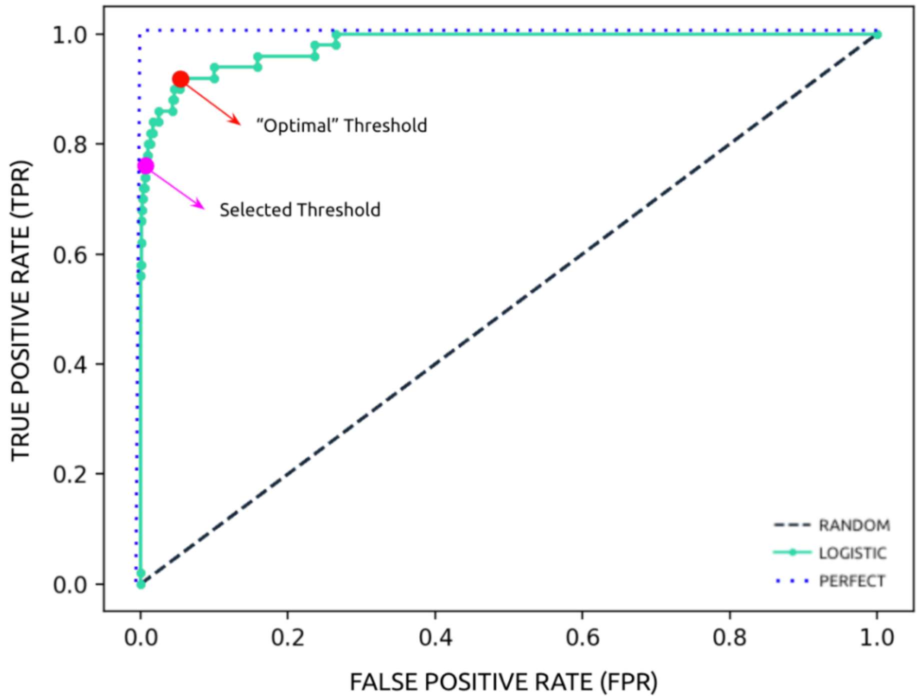


$$FPR = \frac{FP}{FP + TN}$$

- Calculate TPR & FPR for each prob threshold.
- TPR and FPR are positively correlated.
- ROC curve: always start from (0, 0) and end at (1, 1).

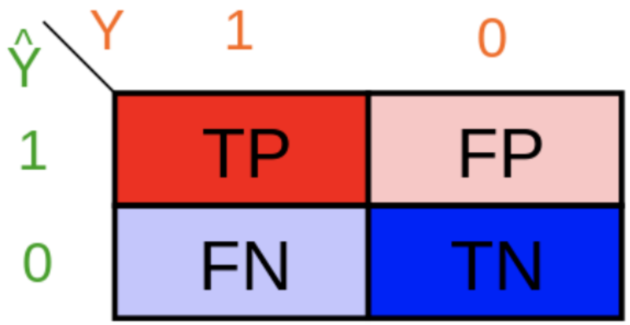
Choose the best threshold for a model

Does the KNN model have a threshold parameters?

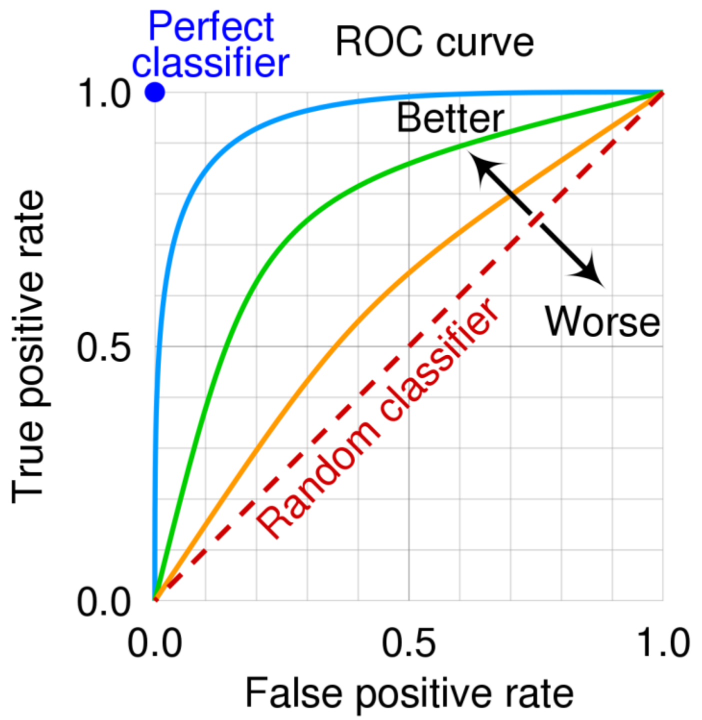


- **Optimal threshold**: maximizing the difference between TPR and FPR;
- **Selected threshold**: highest TPR at the maximum acceptable FPR;
- The selected threshold is often not the optimal threshold.
- *In practice, most models use the default 0.5 for binary classification.*

AUC for model comparison

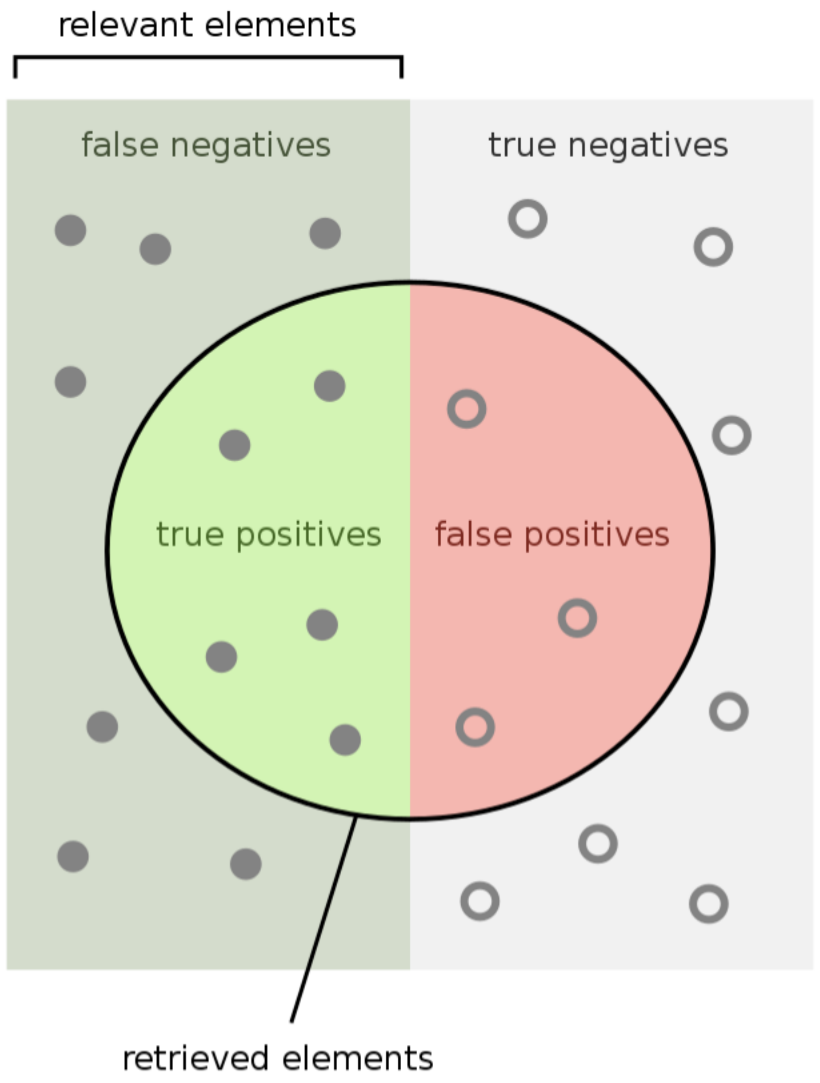


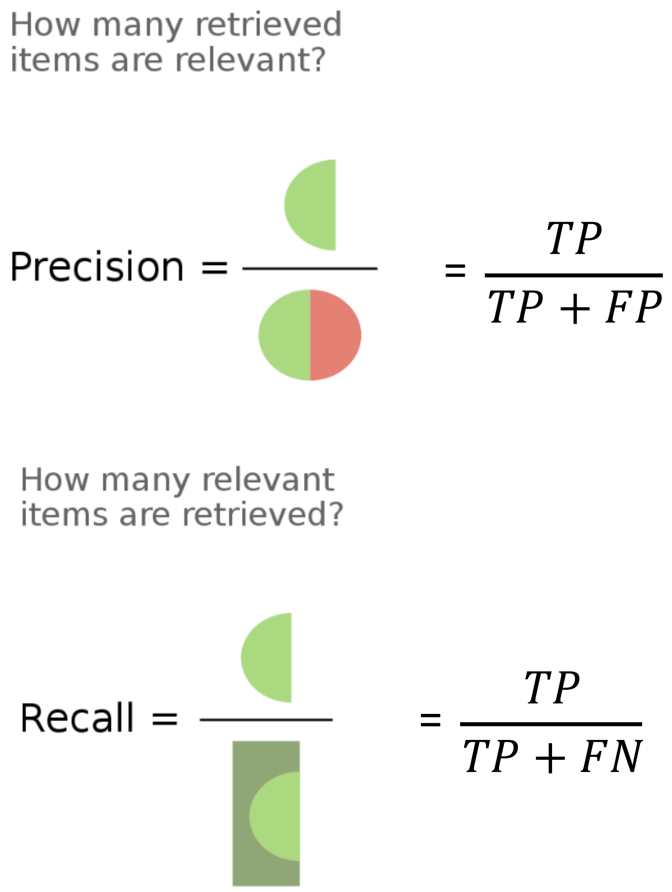
$\hat{Y}$ \ Y	1	0
1	TP	FP
0	FN	TN



- Based on the area under the ROC curve (AUC);
- It's more appropriate when two classes are balanced;
- What about imbalanced datasets? (cancer diagnosis, spam)

Precision vs. recall (focus on minor class)

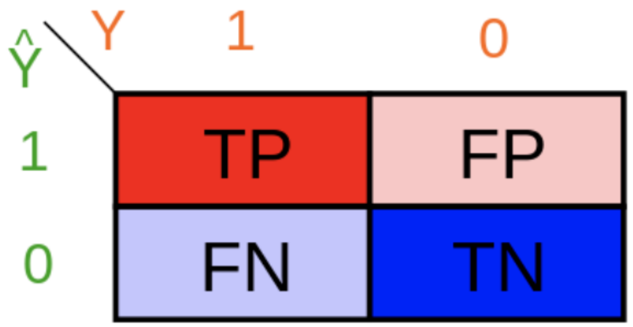



How many retrieved items are relevant?
Precision =
How many relevant items are retrieved?
Recall =

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

Trade-off between precision and recall



$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

- Precision and recall are often inversely related.
- Increasing precision typically decreases recall and vice versa.
- E.g., A model can achieve a high precision by only predicting positive when it is very confident; recall = 1 when always predicting positive.
- *Precision is undefined when only predicting negative.*
- Can plot precision against recall at different thresholds.

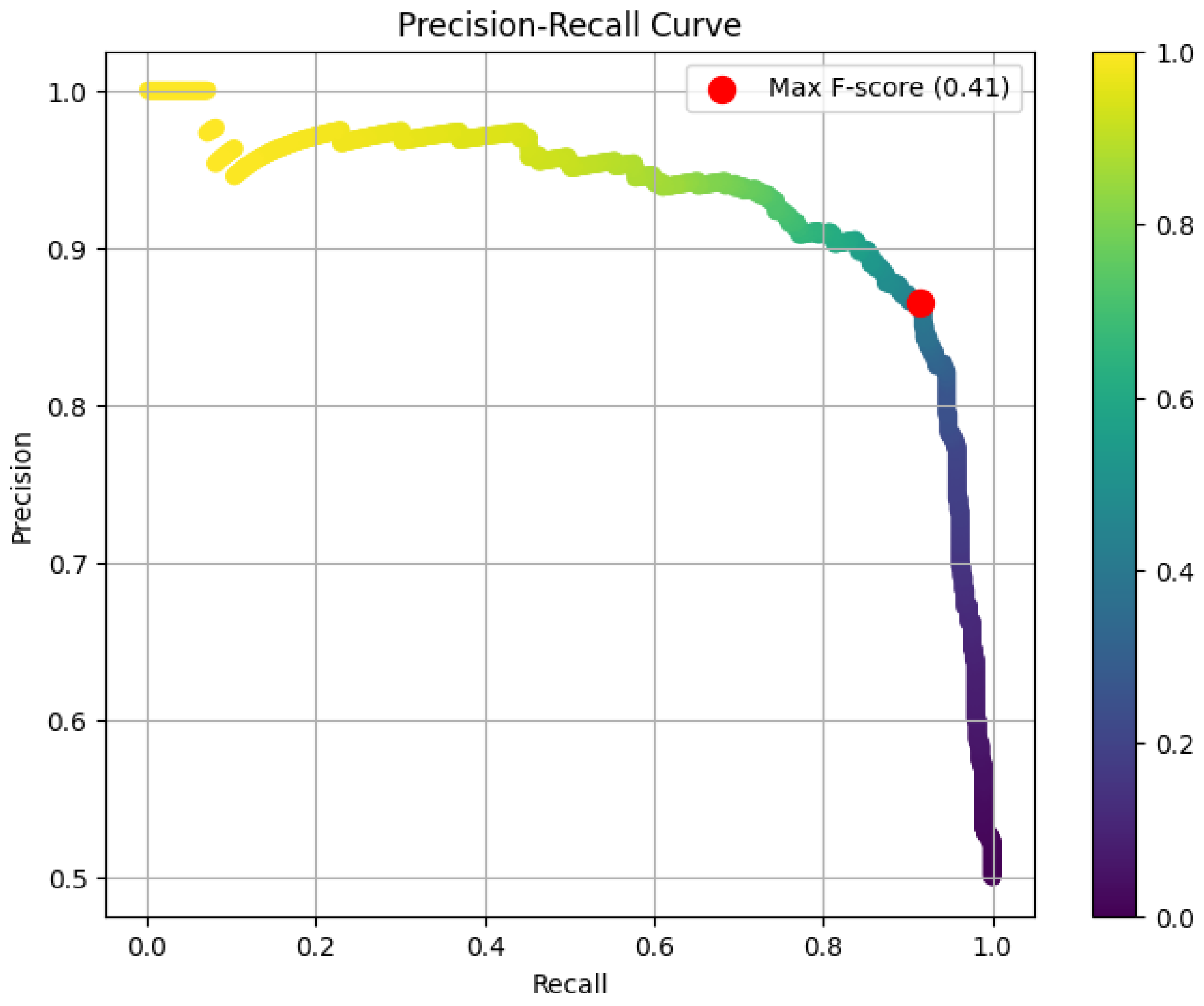
Which metric is more important in cancer diagnosis? Covid-testing?

F_1-score

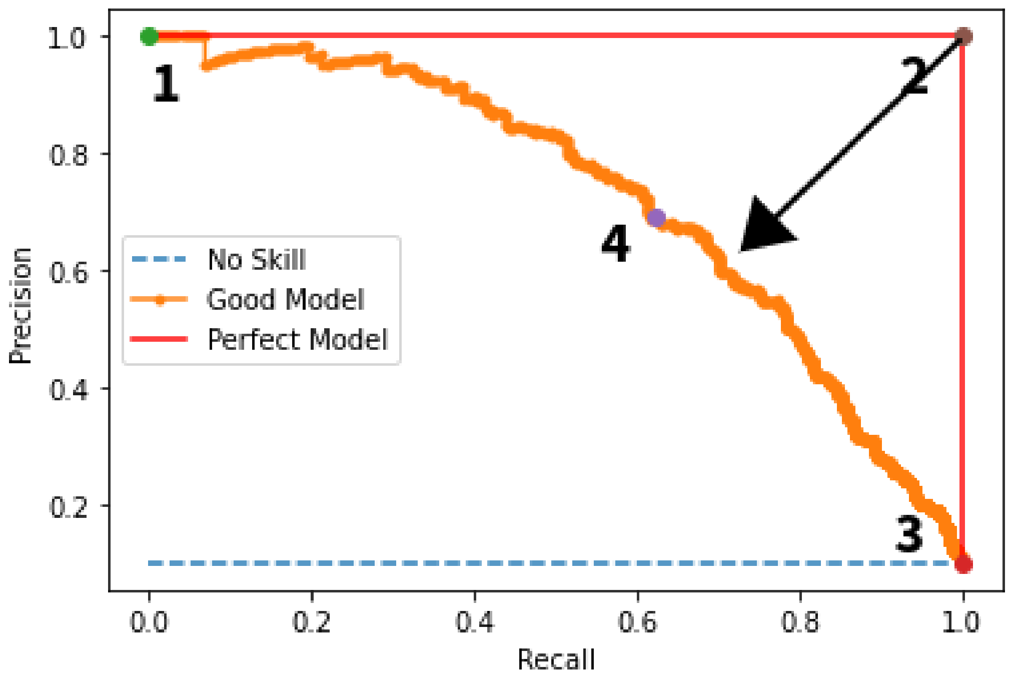
$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2\frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

- F_1 score cares equally about precision and recall;
- Can be used to choose the best threshold for a model to make prediction/classification;

F_1-score for threshold selection



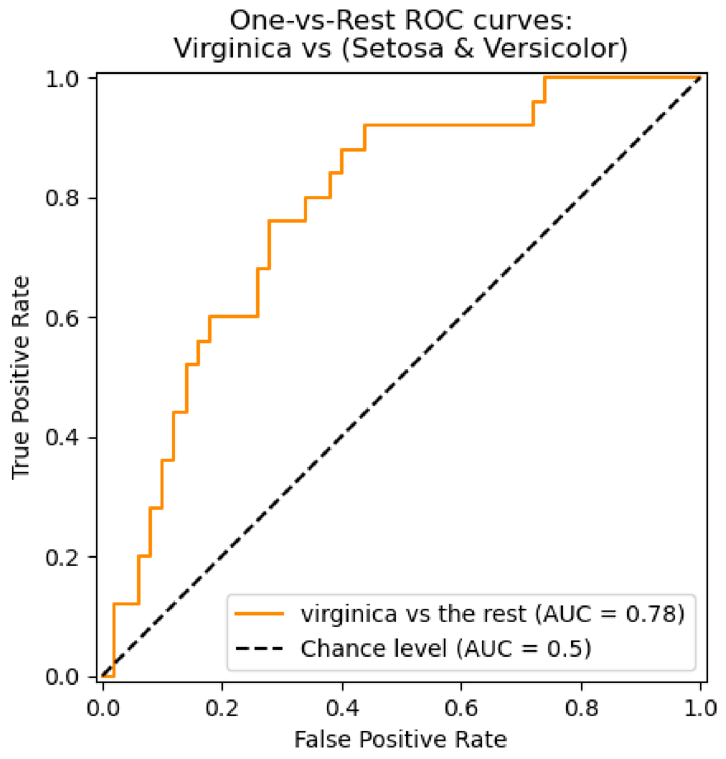
PR curve for model comparison



- A random classifier makes a flat Precision-Recall curve, with precision equal to the proportion of positive instances in the dataset.
- AUC (area under the curve) score closer to 1 means better performance.
- More reliable than ROC for imbalanced datasets.

What about multiclass classification?

- One-vs-rest: compares each class against all others;
- One-vs-one: compares every pairwise classes;



https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html

Summary

- ROC curves summarize the trade-off between the true positive rate and false positive rate for a predictive model using different probability thresholds.
- Precision-Recall curves summarize the trade-off between the true positive rate and the positive predictive value for a predictive model using different probability thresholds.
- ROC curves are appropriate when the observations are balanced between each class, whereas precision-recall curves are appropriate for imbalanced datasets.

Course Notes

- Text analysis next week.
- Unsupervised learning the week after.