

# Data Mining (Week 1)

dm23s1

## Topic 08 : Classification1

### Part 01 : Overview

Preparation

Data Handling

Exploring Data 1

Exploring Data 2

Building Models

Dr Bernard Butler

Department of Computing and Mathematics, WIT.  
(bernard.butler@setu.ie)

Autumn Semester, 2023

Prediction

### Outline

- How classification differs from regression
- Classification metrics
- Logistic regression in practice - Iris dataset
- Logistic regression - how it works

Wrap up

# Data Mining (Week 8)

Introduction

Motivating Example

Preparation

Data Handling

Exploring Data 1

Exploring Data 2

Building Models

Prediction

Regression  
1

Classification  
1

Regression  
2

Classification  
2

Clustering

Wrap up

## Acknowledgment

---

Thanks to Dr Kieran Murphy for some of today's slides.

# Introduction to Classification

## Definition 1 (Classification)

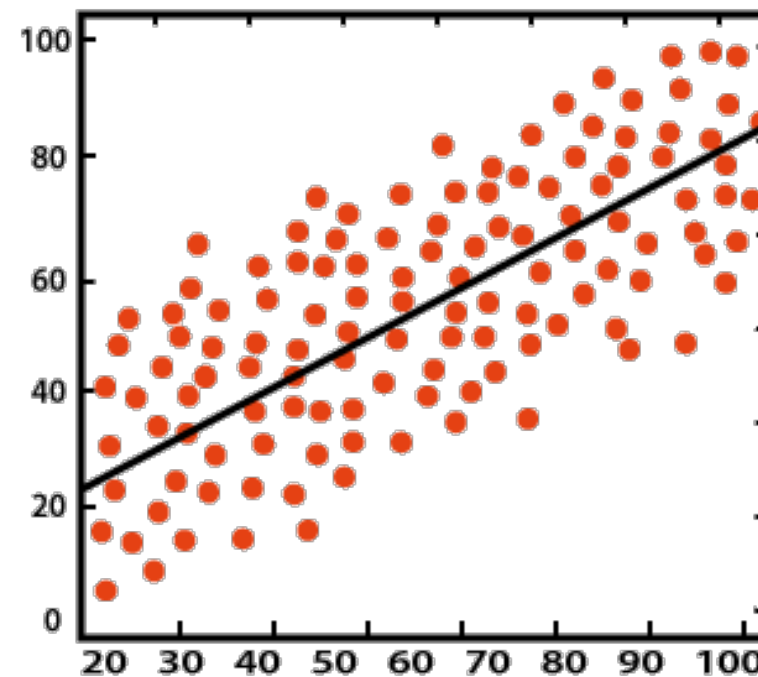
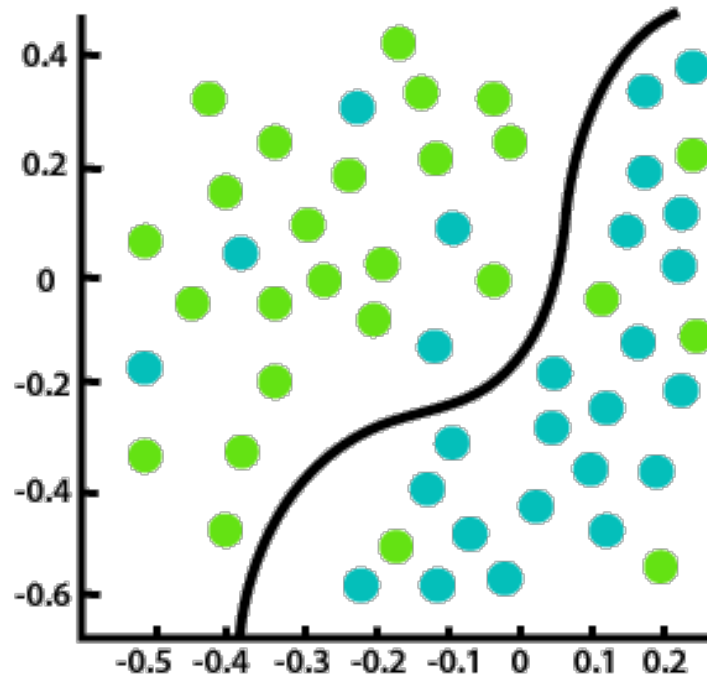
Classification aims to learn a function that takes attribute values and predicts a categorical/qualitative value, such as membership of a class, existence of an effect, etc. The attributes can be categorical or numeric. As with linear regression, classification is an example of supervised learning. It differs from regression because regression predicts a numeric response.

- Some *classifiers* generate class membership probabilities (numeric) en route to predicting class membership (of the most likely class), so the distinction is not always clear-cut.
- There is essentially one regression algorithm (with many variants/enhancements/implementations) but there are *many* classification algorithms.
- You have seen 1 already (KNN) and we introduce another algorithm today.

# Classification vs Regression

**Supervised** data models have a target.

If target is quantitate (continuous) then have a **regression model**, if categorical then **classification model**.



Classification models aim to:



- predict class/label for each new observation,
- define a decision boundary between classes,
- and possibly the probability of being in each class.

Regression models aim to

- predict a continuous value for each new observation.

# Classification vs Regression

- Unlike regression, statistical distributions play a limited role in evaluating a classifier:
  - Scope for hypothesis testing is limited (there is no equivalent of the `statsmodels` diagnostic output (covered in topic 8).
  - Rely on empirical metrics — accuracy, precision, recall, f1-score, auc, ...
- Classification metrics tend to be easier to use/understand than those in regression — classification metrics are based on counts of correct (or incorrect) cases divided by a subset of cases.
- Central concept in classification model is the **confusion matrix**:

		Predicted		
		Negative	Positive	
Actual	Negative	 True Negative (TN)	Type I error False Positive (FP)	$N$
	Positive	Type II error False Negative (FN)	 True Positive (TP)	$P$
		$\hat{N}$	$\hat{P}$	$T$

# Unbalanced Classification Datasets

Practical classification datasets are often **unbalanced** — where the frequency of the classes in the target are very uneven:

- Telecommunication customer churn datasets.
- Credit Card Fraud Detection
- National Institutes of Health Chest X-Ray Dataset

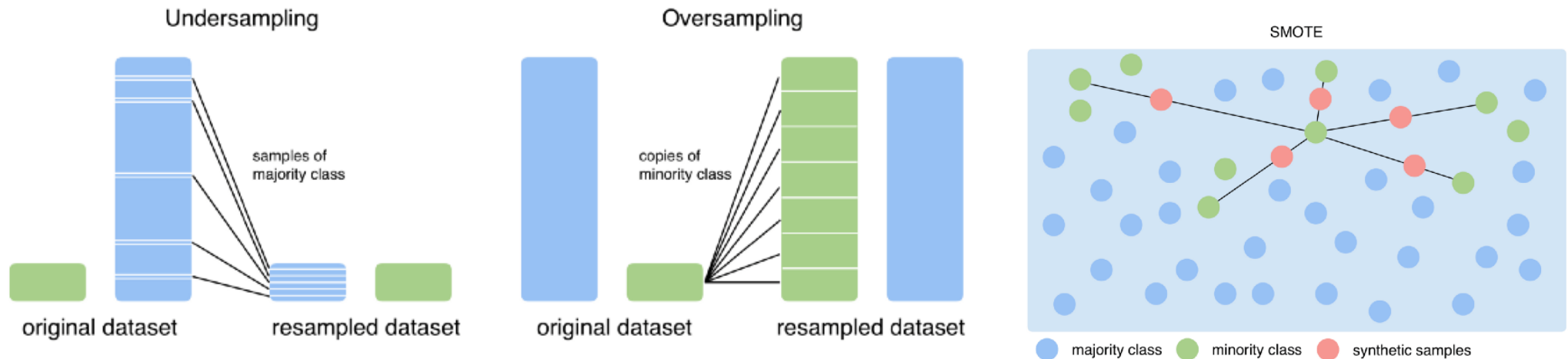
Churn rate of 2%–10%.

0.172% (492 frauds / 284,807 transactions).

14 cases, (size 13 to 3,044) in 5,606 cases

## Solutions

Use suitable metrics and/or



# Summary of Classification Models

Model	Data Pre-processing*		Impact from		Summary
	Normalisation	Scaling	Collinearity	Outliers	
<b>KNN</b>	✓	✓	✓	✗	<ul style="list-style-type: none"> <li>Local approximation, lazy learner</li> <li>Heavy computational requirements</li> </ul>
<b>Logistic Regression</b>	✓	✗	✓	✓	<ul style="list-style-type: none"> <li>Descriptive with good accuracy</li> <li>Reasonable computational requirements</li> </ul>
<b>Naïve Bayes</b>	NA	NA	✓	✗	<ul style="list-style-type: none"> <li>Works with categorical features only</li> <li>Suitable for small train datasets</li> </ul>
<b>Decision Tree</b>	✗	✗	✓	✓	<ul style="list-style-type: none"> <li>Easy to setup and interpret (XAI).</li> <li>Robust to missing data but not to noise.</li> <li>Slow training for larger sets.</li> </ul>
<b>Random Forest</b> (Not this module)	✗	✗	✗	✗	<ul style="list-style-type: none"> <li>High prediction accuracy</li> <li>Limited explainability</li> <li>Works with both continuous and categorical features</li> </ul>
<b>Support Vector Classifier</b> (Not this module)	✗	✗	✗/✓	✓	<ul style="list-style-type: none"> <li>High prediction accuracy</li> <li>Explainability depends on kernel</li> <li>Computational effort depends on kernel</li> </ul>
<b>Neural Networks</b> (Not this module)	✗	✓	✓	✓	<ul style="list-style-type: none"> <li>High prediction accuracy</li> <li>Self-extract features</li> <li>Heavy computational requirements</li> </ul>

\*Use `StandardScaler`, or `RobustScaler` if have outliers.



# Lazy vs Eager Learners

## Lazy learner

Stores training data (or only minor processing) and uses this to compute prediction when given test data.

- Does not generalise until after training
- Does not produce a standalone model
- Training data must be kept for prediction
- Local approximations
- Often based on *search*
- If new data is just added to the training data, it can respond more easily to changing conditions

## Eager learner

Builds a model from the train set, before receiving new data for prediction

- Training has an extra goal: to generalise from the data
- Training has an extra output: standalone model
- Training data can be discarded after use
- Local and/or global approximations
- Based on *computation*
- Models *drift* with time, so not suited to highly dynamic contexts, as it needs retraining

Usually an (eager) model requires much less memory than a (lazy) training set.

# A Non-perfect Test — Type I and Type II Errors

Consider an imperfect test with two outcomes, there are four possible outcomes:

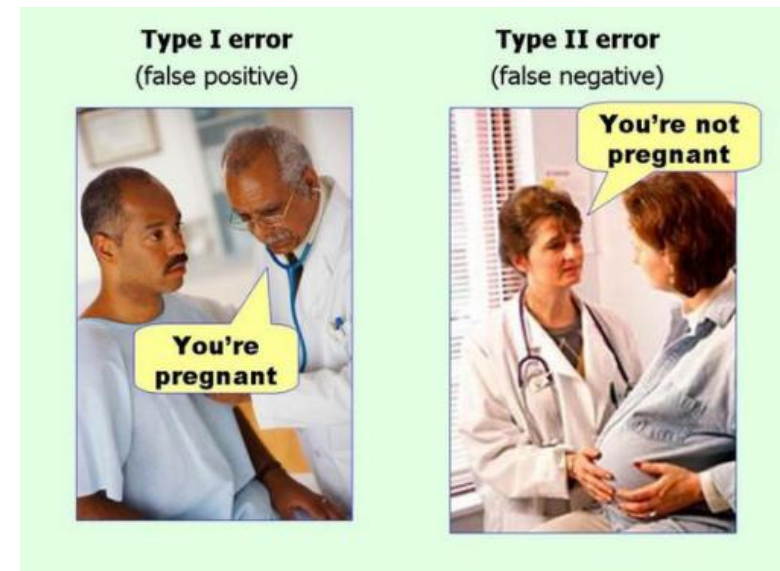
## Confusion Matrix

		Predicted		
		Negative	Positive	
Actual	Negative	<div>✓</div> True Negative (TN)	Type I error False Positive (FP)	N
	Positive	Type II error False Negative (FN)	<div>✓</div> True Positive (TP)	P
		$\hat{N}$	$\hat{P}$	T

- If the test is applied to  $T = P + N = \hat{P} + \hat{N}$  observations / subjects / instances then we have four independent quantities  $TP$ ,  $TN$ ,  $FP$ , and  $FN$ .
- How do we combine these quantities into a single metric ?
- The fraction of correct results seems like a good idea

$$\text{accuracy} = \frac{TP + TN}{P + N}$$

But what happens, if we are testing for an rare event? Maximising accuracy will result in the test always returning negative.



- Ideally we want the probability of either error to be zero but that may not be possible.
- Depending on the conditions we often modify the test to reduce probability of the type of error we don't want at the expense of increasing the probability of the other — think court case vs medical condition.

# Confusion matrix (Contingency table) Metrics

Accuracy — how well model is trained and performs in general

$$\text{Accuracy} = \frac{TP + TN}{P + N}$$

(How often is the classifier correct?)

- False negative rate (FNR) =  $\frac{FN}{P} = 1 - TPR$
- Sensitivity = Recall** = True positive rate (TPR) =  $\frac{TP}{P} = 1 - FNR$   
(Of positive cases that exist how many did we mark positive?)
- Specificity** =  $\frac{TN}{N} = 1 - FPR$   
(When it's actually no, how often does we predict no?)  
(Of cases that are negative, how many did we mark negative?)
- False positive rate (FPR) = false acceptance =  $\frac{FP}{N} = 1 - \text{Specificity}$
- Precision** = positive predictive value (PPV) =  $\frac{TP}{\hat{P}} = \frac{TP}{TP + FP}$   
(Of cases that we marked positive, how many were correct?)

		Predicted		
		Negative	Positive	
Actual	Negative	✓ True Negative (TN)	Type I error False Positive (FP)	N
	Positive	Type II error False Negative (FN)	✓ True Positive (TP)	P
		$\hat{N}$	$\hat{P}$	T

Recall — important when the costs of false negatives are high

Precision — important when the costs of false positives are high

# Confusion matrix (Contingency table) Metrics

## F<sub>1</sub> Score

The F-measure or balanced F-score (F<sub>1</sub> score) is the harmonic mean of precision and recall:

$$F_1 = 2 \left[ \frac{1}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} \right] = 2 \left[ \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \right]$$

## A word of Caution ...

Consider the three binary classifiers A, B and C

	A		B		C	
	T	F	T	F	T	F
T	0.9	0.1	0.8	0	0.78	0
F	0	0	0.1	0.1	0.12	0.1

Metric	A	B	C	(best)
Accuracy	0.9	0.9	0.88	<b>AB</b>
Precision	0.9	1.0	1.0	<b>BC</b>
Recall	1.0	0.888	0.8667	<b>A</b>
F-score	0.947	0.941	0.9286	<b>A</b>

Clearly classifier A is useless since it always predicts label T regardless of the input. Also, B is slightly better than C (lower off- diagonal total).

Yet look at the performance metrics – B is never the clear winner.

We use some metrics because they are easy to understand, and not because they always give the “correct” result.

# Mutual Information is a Better Metric

The **mutual information** between predicted and actual label (case) is defined

$$I(\hat{y}, y) = \sum_{\hat{y}=\{0,1\}} \sum_{y=\{0,1\}} p(\hat{y}, y) \log \frac{p(\hat{y}, y)}{p(\hat{y})p(y)}$$

where  $p(\hat{y}, y)$  is the **joint probability distribution** function.

This gives the intuitively correct rankings  $B > C > A$

Metric	A	B	C
Accuracy	0.9	0.9	0.88
Precision	0.9	1.0	1.0
Recall	1.0	0.888	0.8667
F-score	0.947	0.941	0.9286
<b>Mutual information</b>	<b>0</b>	<b>0.1865</b>	<b>0.1735</b>

# Micro Average vs Macro Average Performance

In a multi-class classifier we have more than two classes. To combine the metrics for individual classes to get an overall system metrics, we can apply either

## Micro-Average Method

Sum up the individual true positives, false positives, and false negatives of the system for different classes and then apply totals to get the statistics.

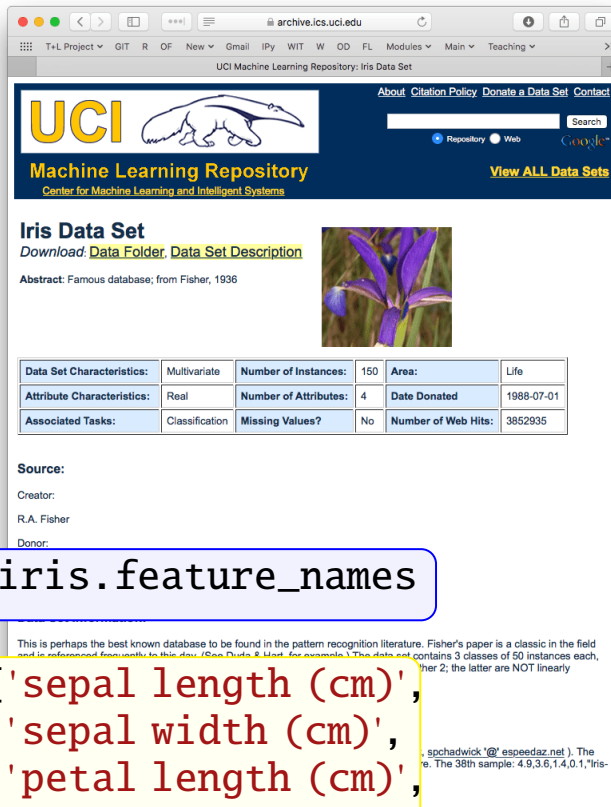
## Macro-average Method

Average the precision and recall of the system on different classes.

See `classification_report` from `sklearn.metrics` (Example: IRIS dataset)

	precision	recall	f1-score	support
setosa	1.00	0.95	0.97	19
versicolor	0.81	0.74	0.77	23
virginica	0.71	0.83	0.77	18
accuracy			0.83	60
macro avg	0.84	0.84	0.84	60
weighted avg	0.84	0.83	0.84	60

# Example: IRIS Dataset — Load



```
from sklearn import datasets
iris = datasets.load_iris()

df = pd.DataFrame(iris.data)
df.columns = iris.feature_names
df['target'] = iris.target_names[iris.target]
df.sample(4)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
2	4.7	3.2	1.3	0.2	setosa
0	5.1	3.5	1.4	0.2	setosa
25	5.0	3.0	1.6	0.2	setosa
70	5.9	3.2	4.8	1.8	versicolor

The data set contains, four numeric features, 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.

## Example: IRIS Dataset — Preprocess Data

We will cover some classifiers in a moment, but for now just treat the classifiers (LogisticRegression) as a black box and focus on the general process:

### Extract the data (features and target)

The IRIS dataset has 4 features, but to simplify visualisation we are only going to use the first two<sup>†</sup> ('sepal length' and 'sepal width'):

```
dataset_name = "IRIS"  
X, y, target_names = iris.data[:, :2], iris.target, iris.target_names
```

### Split dataset into train and test

We will keep 40% of the data for testing. Setting the parameter `random_state` to a value means that we will get a random — but still reproducible — split.

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=666)
```

---

<sup>†</sup>Python for Data Science — Cheat Sheet Numpy Basics



## Example: IRIS Dataset — Fit Model and Predict

### Select classifier

Scikit-learn supports a [large set of classifiers](#), and aims to have a consistent interface to all. First import classifier and create instance ...

```
from sklearn.linear_model import LogisticRegression  
model = LogisticRegression(max_iter=500)
```

### Train model

Then we train (fit) the classifier/model using only the features (`X_train`) and targets (`y_train`) from the train dataset ...

```
model.fit(X_train, y_train)
```

```
LogisticRegression(max_iter=500)
```

### Predict

Now that model is trained, we can use it to generate predictions, using the features (`X_test`) from the test dataset ...

```
y_pred = model.predict(X_test)
```

# Example: IRIS Dataset — Evaluate

## Scoring and confusion matrix

We could just compute the score using whatever metric we have picked ...

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.8333333333333334
```

But this needs context, and even if good can hide critical flaws. Lets look at the confusion matrix ...

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[18, 1, 0],
       [ 0, 17, 6],
       [ 0, 3, 15]])
```

or, to get a nicer output, convert to a DataFrame ...

```
df_cm = pd.crosstab(target_names[y_test], target_names[y_pred])
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
df_cm
```

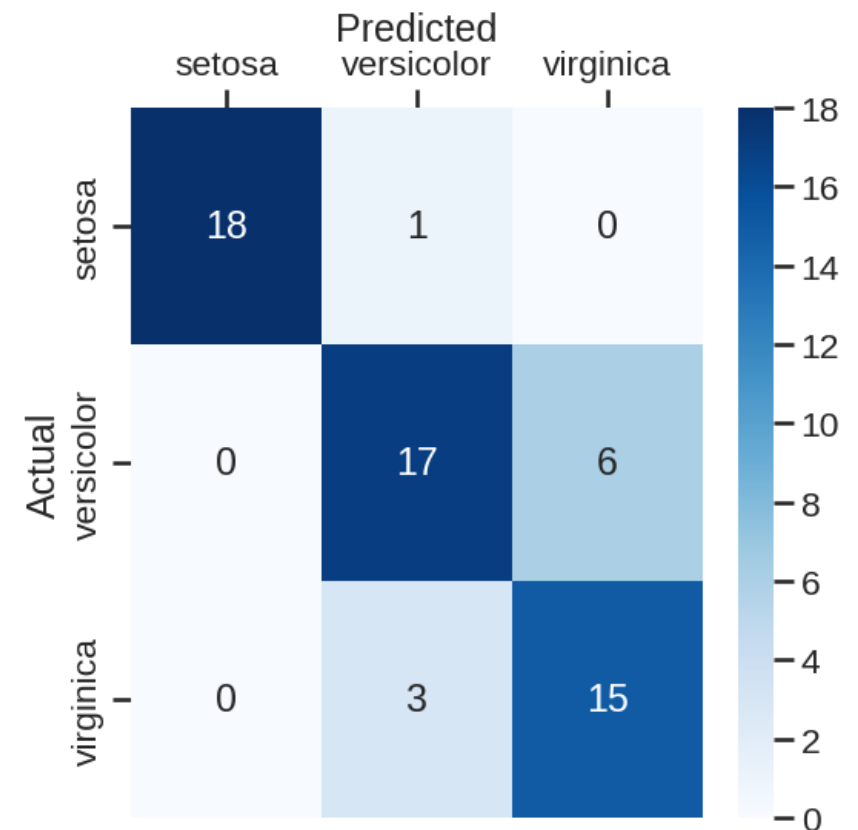
Predicted setosa versicolor virginica			
Actual			
setosa	18	1	0
versicolor	0	17	6
virginica	0	3	15

## Example: IRIS Dataset — Evaluate

The confusion matrix is fundamental in evaluating a classifier, so find a presentation/visualisation that you like and use it. Here I have a heat map representation that I tend to use.

	Predicted setosa versicolor virginica		
Actual			
setosa	18	1	0
versicolor	0	17	6
virginica	0	3	15

The first class **setosa** was only misclassified once, while the classifier had more difficulty between the second two classes.



```
plt.figure(figsize=(6,6))
g = sns.heatmap(df_cm, annot=True, cmap="Blues")
g.xaxis.set_ticks_position("top")
g.xaxis.set_label_position('top')
```

## Example: IRIS Dataset — Evaluate

The classification report, constructed from the confusion matrix, summaries the most common metrics per class and for overall averages ...

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred, target_names=target_names))
```

$$\text{precision (setosa)} = 18 / (18 + 0) = 1$$

$$\text{recall (setosa)} = 18 / (18 + 1) = 0.95$$

**Predicted setosa versicolor virginica**

	<b>Actual</b>		
<b>setosa</b>	18	1	0
<b>versicolor</b>	0	17	6
<b>virginica</b>	0	3	15

	precision	recall	f1-score	support
setosa	1.00	0.95	0.97	19
versicolor	0.81	0.74	0.77	23
virginica	0.71	0.83	0.77	18
accuracy			0.83	60
macro avg	0.84	0.84	0.84	60
weighted avg	0.84	0.83	0.84	60

$$\text{accuracy} = (18 + 17 + 15) / 60 = 0.83$$

$$\text{f1-score (virginica)} = 2 / (1/0.71 + 1/0.83) = 0.77$$

# Motivation

Linear regression is a very powerful and flexible prediction model (Topics 7 and 8)

## Can it be used to predict categorical variables, perhaps coded as numbers?

- Pros

- Linear regression provides a principled way of combining the contributions of the predictors, whether they are numeric or not.
- There is a lot of well-established theory and practice, e.g., in respect of collinearity.
- The model is extremely flexible, e.g., predictors can be nonlinear functions of the features.
- Implementations can use computing resources efficiently.

- Cons

- Prediction is numeric value, needs to be converted to a categorical value.
- Conversion function introduces unwelcome features, e.g., ordering and scaling, that do not apply to nominal variables.
- Interpretation of linear regression in terms of classification performance is tricky because the conversion function needs to balance continuity against evaluating to either 0 or 1.

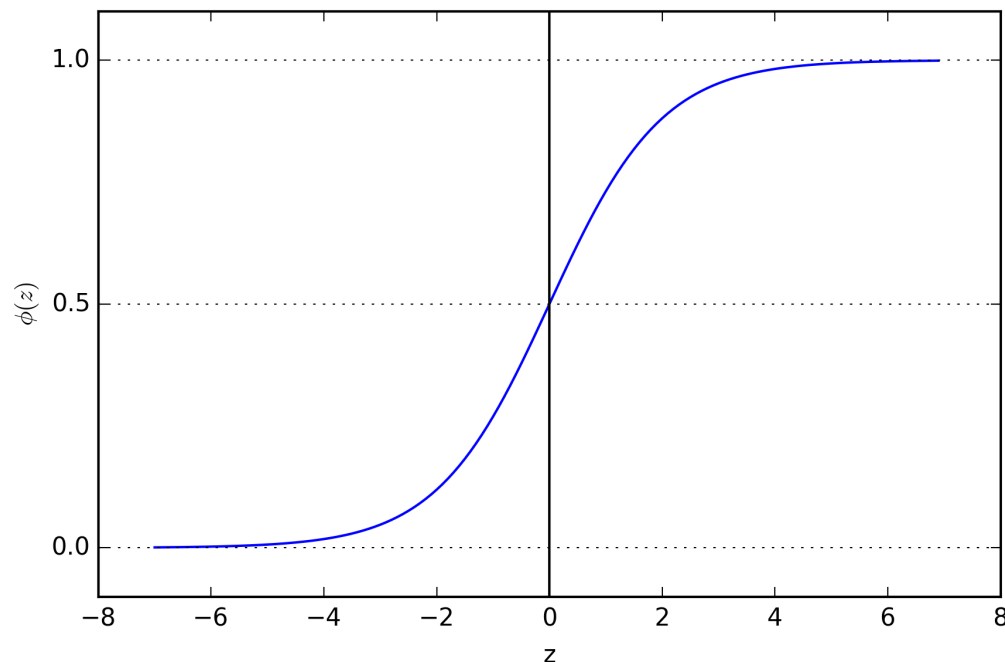
So—is there any hope for using linear regression in classification?

# Probability view of Classification

**When we predict a nominal value, it is equivalent to saying that the probability that a given observation takes that value is high and that it takes any other value is low.**

- Probabilities are numeric, but their range is restricted to  $[0, 1]$ .
- We need a function of the predictors with the following properties
  - it has the range  $[0, 1]$
  - it is defined over a domain which is  $(-\infty, \infty)$
  - it can evaluate to 0 or 1, depending on its input.
- A line (like  $y = \beta_0 + \beta_1 x$ ), such as we used in previous topics, does not have these properties
- Ideally, the function should be smooth and well-behaved everywhere, *and* evaluate to 0 or 1 as appropriate.
- Note that the classes are labeled as 0 or 1 (binary classification).

# Introducing the logistic function



## Definition 2 (logistic function)

The curve above is given by the logistic function, which can be written as  $p(z) = \frac{e^z}{1+e^z}$ . Letting  $z = \beta_0 + \beta_1 X$ , we have  $p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$  for some  $\beta_0$  and  $\beta_1$ .

## Predicting the (log) odds ratio

$$\begin{aligned}
 p(X) &= \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} & (1) \\
 \implies e^{\beta_0 + \beta_1 X} &= p(X) + e^{\beta_0 + \beta_1 X} p(X) \\
 \implies (1 - p(X))e^{\beta_0 + \beta_1 X} &= p(X) \\
 \implies \frac{p(X)}{1 - p(X)} &= e^{\beta_0 + \beta_1 X} \\
 \implies \log \left( \frac{p(X)}{1 - p(X)} \right) &= \beta_0 + \beta_1 X & (2)
 \end{aligned}$$

The expression on the left of (2) is called the *log-Odds Ratio* (or  $\text{logit}(X)$ ). When  $p(X) \rightarrow 1$ ,  $\text{logit}(X) \rightarrow \infty$  and when  $p(X) \rightarrow 0$ ,  $\text{logit}(X) \rightarrow -\infty$ , as required.

The expression on the right of (2) is a linear form in  $\beta$ , as used in linear regression.

For **training**: Compute  $\text{logit}(X)$  from the training labels and use linear regression to get  $\{\beta_i\}$ .

For **prediction**: Substitute  $X$ ,  $\{\beta_i\}$  in the logistic function (1) to obtain class  $\text{round}(p(X))$ .



# Logistic regression summary

## Training

Given (labeled) training data, convert the label to the equivalent logit value and use *maximum likelihood estimation* to look for the parameters  $\beta$  that make the observed data as likely as possible. Extension to multiple predictors is trivial.

## Prediction

Given the fitted  $\beta$ , just evaluate the logistic function for a specific  $X$ . The resulting  $p(X)$  will hopefully be near 0 or 1 and can be interpreted according to how “success” ( $p = 1$ ) is defined.

## Extension to categorical predictors

As with linear regression: create dummy (binary) indicator (0,1)-valued variables, one for each level of the categorical predictor.

## Extension to non-binary predicted values

We can convert to extra indicator variables, but at some cost in complexity. Therefore, logistic regression is best suited to binary prediction.

# Logistic regression in python

Python's `scikit-learn` and `statsmodel` libraries provide a general interface to model fitting that abstracts away logistic functions and other details.

## Method (Recognising the Handwritten Digits)

```
1 from sklearn.linear_model import LogisticRegression
2
3 # Get and configure a LogisticRegression object, with an L2 regularisation penalty
4 clf = LogisticRegression(penalty='l2')
5
6 # Fit the training data
7 clf.fit(Xtrain, ytrain)
8
9 # Using the beta parameters that have just been learned and are in clf, predict (recognise) the test data
10 ypred = clf.predict(Xtest)
```

# Using Categorical Features in (Logistic) Regression

How can Categorical-valued features participate in linear models?

## The Problem

Given the following fragment of a dataset, where the goal is to predict the salary of employees in a large organisation:

```
df = pd.read_csv('data/team.csv',\n                 index_col="Name")\ndf
```

yielding

Role Skilled Salary			
Name			
<b>Alice</b>	Designer	Yes	40000
<b>Bob</b>	Programmer	No	25000
<b>Carol</b>	Tester	No	30000

How can this participate in a regression model, where all quantities take numeric values?

## Using pandas .getdummies() on a binary-valued column

```
dfSkilledDummies = pd.get_dummies(df['Skilled'],\n    prefix='Skilled',\n    dtype=int)\ndfSkilledDummies
```

	Skilled_No	Skilled_Yes
Name		
Alice	0	1
Bob	1	0
Carol	1	0

Note that a binary-valued column becomes 2 dummy columns

## Reducing redundancy (by 1) in 2 dummy columns

```
dfSkilledIndicators = pd.get_dummies(df['Skilled'],\
    prefix='Skilled',\
    drop_first=True,\
    dtype=int)\
    .rename(columns={"Skilled_Yes": "IsSkilled"})
dfSkilledIndicators
```

IsSkilled	
Name	
Alice	1
Bob	0
Carol	0

➤ A single indicator column can replace a group of 2 dummy columns

## Using pandas .getdummies() on a multi-valued column

```
dfRoleDummies = pd.get_dummies(df['Role'],\n                                prefix='Role',\n                                dtype=int)\ndfRoleDummies
```

Note that an  $n$ -valued column becomes  $n$  dummy columns

	Role_Designer	Role_Programmer	Role_Tester
Name			
Alice	1	0	0
Bob	0	1	0
Carol	0	0	1

# Reducing redundancy (by 1) in $n$ dummy columns

```
dfRoleIndicators = pd.get_dummies(df['Role'],\
    prefix='Role',\
    drop_first=True,\
    dtype=int)\
    .rename(columns={\
        "Role_Programmer": "IsProgrammer",\
        "Role_Tester": "IsTester"})
```

dfRoleIndicators

	IsProgrammer	IsTester
Name		
Alice	0	0
Bob	1	0
Carol	0	1

$n - 1$  indicator columns can replace a group of  $n$  dummy columns

# Replacing categorical solumns by their dummy equivalents

```
dfAllDummies = df.drop(["Role", "Skilled"],\
                        axis=1)\
                    .join(dfRoleDummies, on='Name')\
                    .join(dfSkilledDummies, on='Name')\
dfAllDummies
```

	Salary	Role_Designer	Role_Programmer	Role_Tester	Skilled_No	Skilled_Yes
Name						
Alice	40000	1	0	0	0	1
Bob	25000	0	1	0	1	0
Carol	30000	0	0	1	1	0

Categorical columns have been encoded to numerical columns

Can now use regression-type approaches, but need more features in the dataframe



# Review of Regression 1

---

- Classification is one of the most common machine learning tasks
- For regression, the focus is on residuals, for classification it is on the confusion matrix
- Performance metrics are based on ratios and independent of the algorithm used
- Trade-offs are needed: Type 1 vs Type 2 errors and associated classification metrics
- Builds upon all the existing EDA, model building and multivariate analysis we saw before

Next topic - we introduce 2 new classification algorithms