Classification with Two Classes

* Will Johnny like or dislike the pie?
* Training data:

```
Johnny likes:
- Round pie
- Square pie
- Triangle pie
Johnny does NOT like:
- Circular pie
- Rectangular pie
- Triangular pie
```

* Features: Shape (round, square, triangle), filling (white, gray, dark), crust (thick, thin, light, dark), size (big, small) etc
* To make prediction match features of pie against previous examples from training data
Classification with Two Classes

* Examples:
  * Movies reviews: positive or negative?
  * Images: Does a picture contain human faces?
  * Finance: Is person applying for a loan likely to repay it?
  * Advertising: If display an ad on web page is the person likely to click on it?
  * Online transactions: fraudulent or not?
  * Tumor: malignant or benign?

* As before $x=$ “input” variable/features e.g. text of email, location, nationality

* Now $y=$ “output” variable/“target” variable only takes values -1 or 1 (with linear regression $y$ was real valued). In classification $y$ often referred to as the label\(^1\).

* We want to build a classifier that predicts the label of a new object e.g. whether a new email is spam or not.

\(^1\)Note could also use values 0 and 1 rather than -1 and 1, leave this as an exercise
As before $\theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$ with $x_0 = 1$, $x_1, \ldots, x_n$ the input features and $\theta_0, \ldots, \theta_n$ the (unknown) parameters.

Model: $\text{sign}(\theta^T x)$ i.e. predict output $+1$ when $\theta^T x > 0$ and output $-1$ when $\theta^T x < 0$. Decision boundary is $\theta^T x = 0$ (green line in plot above)

$\theta^T x = 0$ defines a point in one dimension e.g.
$1 + 0.5 x_1 = 0 \rightarrow x_1 = -2$ ...

... a line in two dimensions e.g.
$2 + x_1 + 2x_2 = 0 \Rightarrow x_2 = -x_1/2 - 1$ ...

.. and a plane in higher dimensions
Logistic Regression: Decision Boundary

* Example: suppose $x$ is vector $x = [1, x_1, x_2]^T$ e.g. $x_1$ might be tumour size and $x_2$ patient age.

* $\theta_0 = 0$, $\theta_1 = 0.5$, $\theta_2 = -0.5$.
* $\text{sign}(\theta^T x) = +1$ when $0.5x_1 - 0.5x_2 > 0$ i.e. when $x_1 > x_2$.
* When data can be separated in this way we say that it is \textit{linearly separable}.
* Often the 3D plot on left is sketched in 2D as shown in right (easier to draw!)
Not all data is linearly separable e.g.
Logistic Regression: Choice of Cost Function

* Training data: \(\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}\)

* \(x \in \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}\), \(x_0 = 1\), \(y \in \{-1, 1\}\)

* Model: \(h_\theta(x) = \text{sign}(\theta^T x)\)

* How to choose parameters \(\theta\)?
Logistic Regression: Choice of Cost Function

Model: \( \text{sign}(\theta^T x) \) i.e. predict output +1 when \( \theta^T x > 0 \) and output -1 when \( \theta^T x < 0 \)

Suppose we try square error \( \frac{1}{m} \sum_{i=1}^{m} (h_{\theta}(x^{(i)}) - y^{(i)})^2 \).

- Roughly speaking, minimising square error fits the same line to the data that we would fit by eye.
- Works ok on left-hand plot above: \( \text{sign}(\theta^T x) = -1 \) for points to the left of green line and \( \text{sign}(\theta^T x) = +1 \) for points to the right.
- Not so good in right-hand plot - data points far to the right pull point where \( \theta^T x \) crosses zero to the right, so causing misclassification.
* We might consider the **0-1 loss function:**

\[
\frac{1}{m} \sum_{i=1}^{m} \mathbb{I}(h_{\theta}(x^{(i)}) \neq y^{(i)})
\]

where indicator function \( \mathbb{I} = 1 \) if \( h_{\theta}(x^{(i)}) \neq y^{(i)} \) and \( \mathbb{I} = 0 \) otherwise. But hard to work with.

* For logistic regression we use:

\[
\frac{1}{m} \sum_{i=1}^{m} \log \left( \frac{1 + e^{-y^{(i)} \theta^T x^{(i)}}}{\log(2)} \right)
\]

noting that \( y = -1 \) or \( y = +1 \). Scaling by \( \log(2) \) is optional, but makes the loss 1 when \( y^{(i)} \theta^T x^{(i)} = 0 \).
Logistic Regression: Choice of Cost Function

Loss function: \( \log(1 + e^{-y\theta^T x}) / \log(2) \)

- So a small penalty when \( \theta^T x \gg 0 \) and \( y = 1 \), and when \( \theta^T x \ll 0 \) and \( y = -1 \).
- Minimising this thus gives preference to \( \theta \) values that push \( \theta^T x \) well away from the decision boundary \( \theta^T x = 0 \).
Summary

- Model: $h_\theta(x) = \text{sign}(\theta^T x)$
- Parameters: $\theta$
- Cost Function: $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \log(1 + e^{-y^{(i)} \theta^T x^{(i)}})$
- Optimisation: Select $\theta$ that minimises $J(\theta)$
Gradient Descent

As before, can find $\theta$ using gradient descent. For $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \log(1 + e^{-y^{(i)} \theta^T x^{(i)}})$:

\[
\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} -y^{(i)} x_j^{(i)} \frac{e^{-y^{(i)} \theta^T x^{(i)}}}{1+e^{-y^{(i)} \theta^T x^{(i)}}}
\]

(Remember $\frac{d \log(x)}{dx} = \frac{1}{x}$, $\frac{d \exp(x)}{dx} = \exp(x)$ and chain rule $\frac{df(z(x))}{dx} = \frac{df}{dz} \frac{dz}{dx}$)

So gradient descent algorithm is:

* Start with some $\theta$
* Repeat:
  * for $j=0$ to $n$ \{ $\delta_j := -\frac{\alpha}{m} \sum_{i=1}^{m} y^{(i)} x_j^{(i)} \frac{e^{-y^{(i)} \theta^T x^{(i)}}}{1+e^{-y^{(i)} \theta^T x^{(i)}}}$ \}
  * for $j=0$ to $n$ \{ $\theta_j := \theta_j + \delta_j$ \}
* $J(\theta)$ is convex, has a single minimum. Iteration moves downhill until it reaches the minimum
Logistic regression:

```python
import numpy as np
Xtrain = np.random.uniform(0, 1, 100)
ytrain = np.sign(Xtrain-0.5)
Xtrain = Xtrain.reshape(-1, 1)
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(penalty='none', solver='lbfgs')
model.fit(Xtrain, ytrain)
print("intercept %f, slope %f"%(model.intercept_, model.coef_))
```

Typical output:
intercept -267.026565, slope 529.954559

* Prediction $\hat{y} = \text{sign}(-267.026565 + 529.954559x)$
* i.e. $y = +1$ when $-267.026565 + 529.954559x > 0$ and $y = -1$ when $-267.026565 + 529.954559x < 0$
* i.e. $y = +1$ when $x > 267.026565/529.954559 = 0.50392$ and $y = -1$ when $x < 267.026565/529.954559 = 0.5039$
* We generated data using $y = +1$ when $x > 0.5$ and $y = -1$ when $x < 0.5$. So model learned from training data is roughly correct, but not perfect. That’s normal. Why?
Plotting predictions

```python
import matplotlib.pyplot as plt
plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
plt.scatter(Xtrain, ytrain, color='red', marker='+')
plt.scatter(Xtrain, ypred, color='green', marker='+')
plt.xlabel("input x"); plt.ylabel("output y")
plt.legend(['train', 'predict'])
plt.show()
```
Examples:

- Email folder tagging: work, friends, family, hobby
- Weather, sunny, cloudy, rain, snow
- Given where I live in Dublin, predict which political party I’ll vote for.

Now $y$ = “output” variable/“target” variable takes values 0, 1, 2, .... E.g. $y = 0$ if sunny, $y = 1$ if cloudy, $y = 2$ if rain etc.
Train a classifier $\text{sign}(\theta^T x)$ for each class $i$ to predict the probability that $y = i$. Training data: re-label data as $y = -1$ when $y \neq i$ and as $y = 1$ when $y = i$, so we’re back to a binary classification task.
In an SVM use the “hinge” loss function $\max(0, 1 - y\theta^T x)$:

Main differences from logistic loss function:

* hinge-loss is not differentiable ("non-smooth")
* hinge loss assigns zero penalty to all values of $\theta$ which ensure $\theta^T x \geq 1$ when $y = 1$, and $\theta^T x \leq -1$ when $y = -1
In an SVM use the “hinge” loss function \( \max(0, 1 - y\theta^T x) \):

* So long as \( y\theta^T x > 0 \) then by scaling up \( \theta \) sufficiently, e.g. to \( 10\theta \) or \( 100\theta \), then we can always force \( y\theta^T x > 1 \) i.e. \( \max(0, 1 - y\theta^T x) = 0 \)

* To get sensible behaviour we have to penalise large values of \( \theta \). We do this by adding penalty \( \theta^T \theta = \sum_{j=1}^{n} \theta_j^2 \)

* Final SVM cost function is:

\[
J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y^{(i)} \theta^T x^{(i)}) + \theta^T \theta / C
\]

where \( C > 0 \) is a weighting parameter that we have to choose (making \( C \) bigger makes penalty less important).
SVM Summary

* Model: \( h_\theta(x) = \text{sign}(\theta^T x) \)
* Parameters: \( \theta \)
* Cost Function: \( J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y^{(i)} \theta^T x^{(i)}) + \frac{\theta^T \theta}{C} \)
* Optimisation: Select \( \theta \) that minimise \( J(\theta) \)
* Observe that only difference from Logistic Regression is in the choice of cost function.
* SVM cost function:
  * Includes penalty \( \frac{\theta^T \theta}{C} \). We can also add a penalty like this to Logistic Regression though \( \rightarrow \) regularisation, we’ll come back to this later
  * Terms in sum are zero for points where \( y^{(i)} \theta^T x^{(i)} \geq 1 \rightarrow \) this is the important difference.
    * It means that training data points \( (x^{(i)}, y^{(i)}) \) with \( y^{(i)} \theta^T x^{(i)} \geq 1 \) don’t contribute to the cost function
    * Only the training data points with \( y^{(i)} \theta^T x^{(i)} < 1 \) are relevant \( \rightarrow \) support vectors. Can make computations more efficient.
Subgradient descent algorithm for SVMs is:

1. Start with some $\theta$
2. Repeat:
   a. for $j=0$ to $n$
      * $\delta_j := -\alpha(2\theta_j/C - \frac{1}{m} \sum_{i=1}^{m} y^{(i)} x_j^{(i)} \mathbb{1}(y^{(i)} \theta^T x^{(i)} \leq 1))$
   b. for $j=0$ to $n$ \{ $\theta_j := \theta_j + \delta_j$ \}
3. where $\mathbb{1}(y^{(i)} \theta^T x^{(i)} \leq 1) = 1$ when $y^{(i)} \theta^T x^{(i)} \leq 1$ and zero otherwise.

$J(\theta)$ is convex, has a single minimum. Iteration moves downhill until it reaches the minimum.
Logistic regression:

```python
from sklearn.svm import LinearSVC
model = LinearSVC(C=1.0).fit(Xtrain, ytrain)
print("intercept %f, slope %f"%(model.intercept_, model.coef_))
```

**Typical output:**

```
intercept -1.890453, slope 3.867570
```

* So prediction is $y = +1$ when $x > 1.890453/3.867570 = 0.4888$ and $y = -1$ when $x < 0.4888$

* cf Logistic Regression: intercept -267.026565, slope 529.954559 i.e. $y = +1$ when $x > 0.5039$ and $y = -1$ when $x < 0.5039$

* Recall penalty term encourages SVM to choose smaller $\theta$. Changing to use $C = 1000$ gives intercept -19.830632, slope 40.271028 i.e. decision boundary $x = 19.830632/40.271028 = 0.4924$