Lecture14-DS4400

k-Nearest Neighbors (k-NN) & Logistic Regression

Miguel Fuentes-Cabrera

Fall 2025

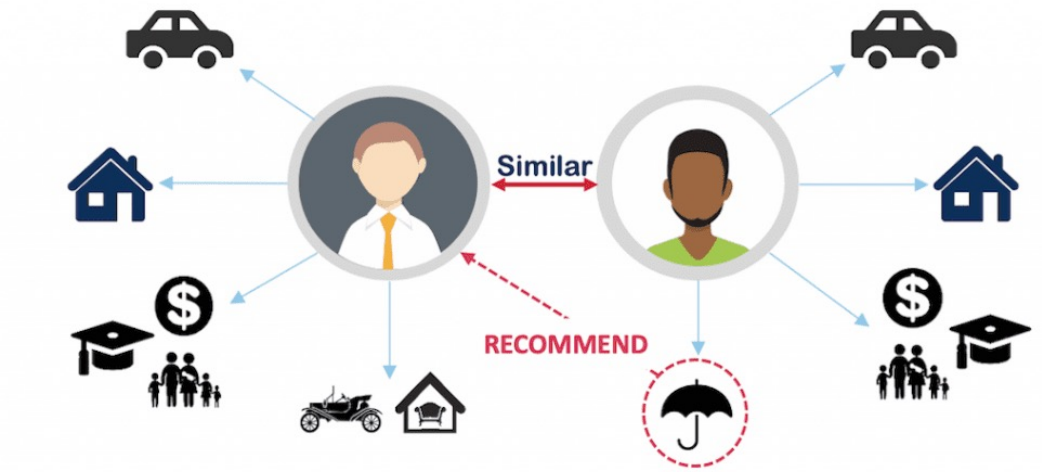
Northeastern University

Today's Agenda

k-NN & PCA

- Recommendation System
- Instance Based Learning
- k Nearest Neighbors (kNN)
- Logistic Regression

Recommendation Systems

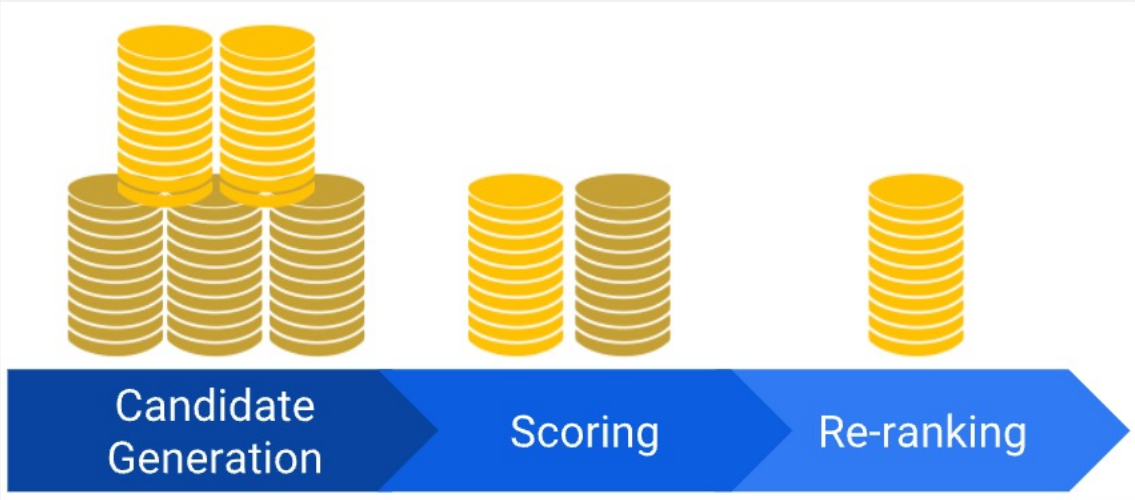


Recommendation Systems

Interested to learn more?

- https://developers.google.com/machine-learning/recommendation
- https://www.datacamp.com/tutorial/recommender-systems-python

A common architecture for recommendation systems:



Credit: https://developers.google.com/machine-learning/recommendation/overview/types

Type	Definition	Example
Content-based filtering	Uses similarity between items to recommend items similar to what the user likes.	If user A watches two cute cat videos, then the system can recommend cute animal videos to that user.
Collaborative filtering	Uses similarities between queries and items simultaneously to provide recommendations.	If user A is similar to user B, and user B likes video 1, then the system can recommend video 1 to user A (even if user A hasn't seen any videos similar to video 1).

Case-Study
(University Track & Field Team)

Every year Northeastern University recruits some additional talented athletes to join the team.

- **How kNN can support our team selection process?**

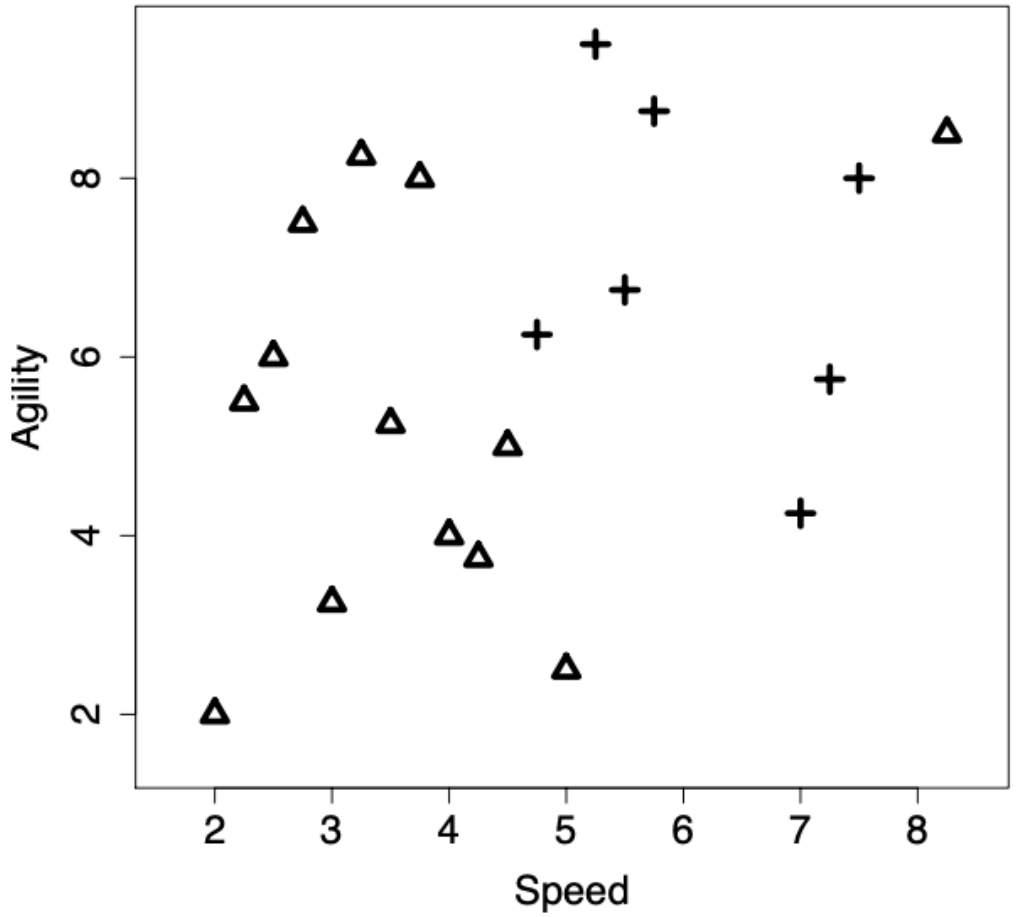
Fundamentals (K-NN)

The fundamentals of similarity-based learning are:	
Feature space	**Similarity metrics**

Fundamentals (Feature Space)

Figure: A feature space plot of The speed and agility ratings for 20 college athletes labelled with the decisions for whether they were drafted or not.

The Triangles represent 'Non-draft' instances and the crosses represent the *'Draft'* instances.



Fundamentals (Feature Space)

A feature space is an abstract n-dimensional space

- In a feature space, each **descriptive feature** corresponds to an axis.
- Each **instance** in the dataset is mapped to a point in the feature space

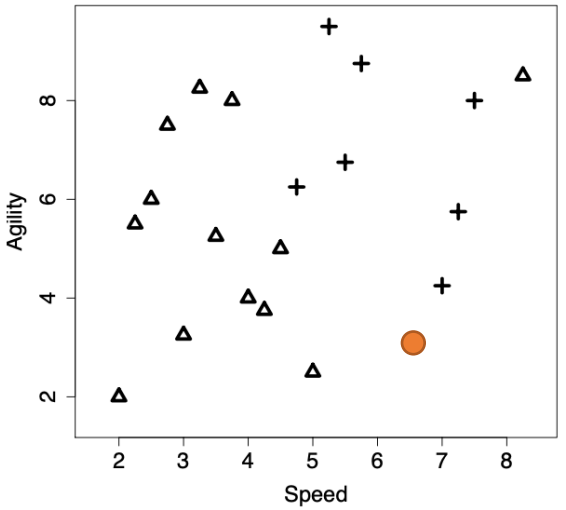
K Nearest Neighbours (K-NN)

Example

- Should we draft an athlete with the following profile:

Query: SPEED = 6.75, AGILITY= 3

Using 1-NN?

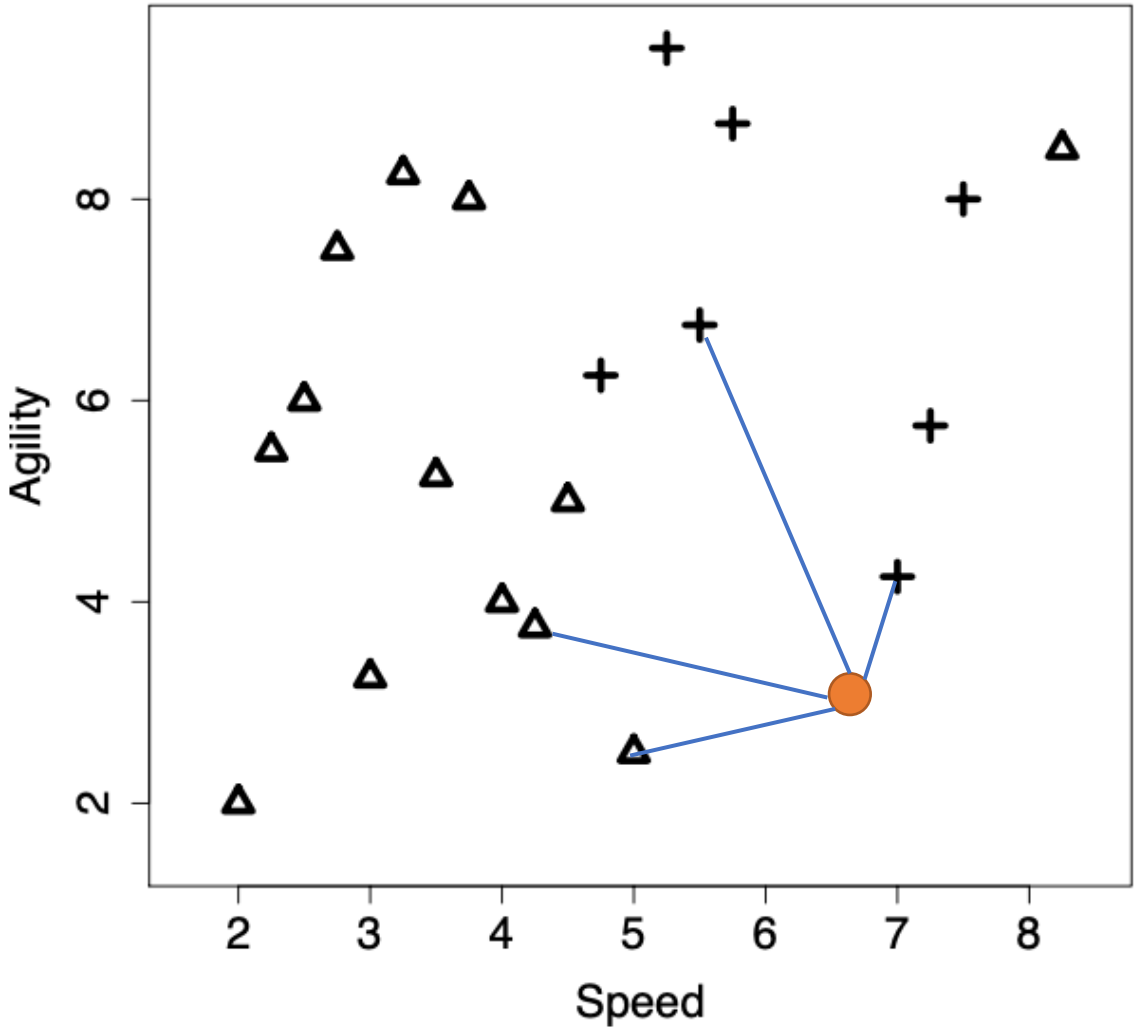


K Nearest Neighbours (K-NN) – Exercise 1

Question: Should we select an athlete with the following profile this year?

Query: SPEED = 6.75, AGILITY= 3

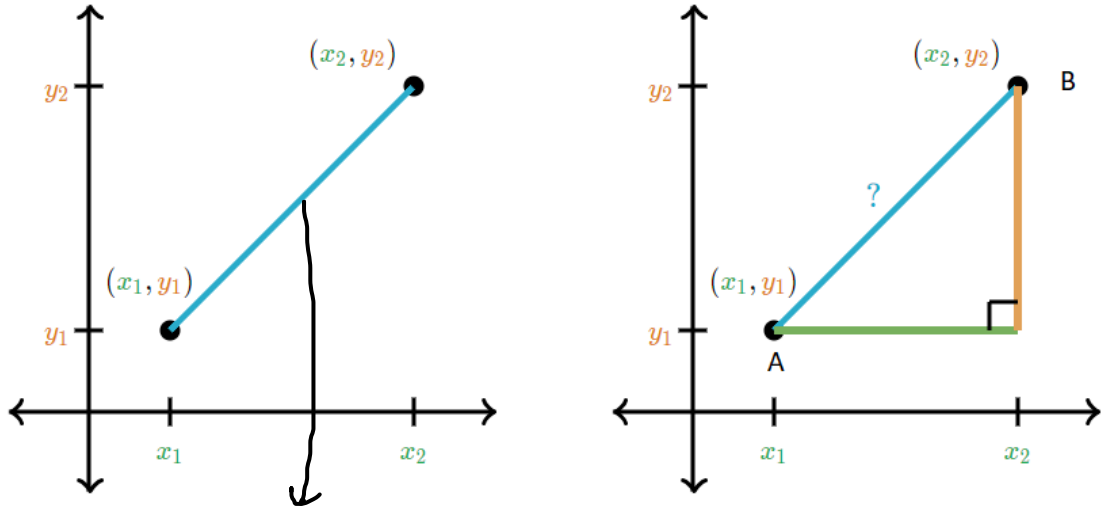
Using 1-NN?



Fundamentals (Distance Metrics)

One of the best known metrics is Euclidean distance which computes the length of the straight line between two points. Euclidean distance between two instances **a** and **b** in a *m*-dimensional feature space is defined as:

$$Euclidean(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{m} (\mathbf{a}[i] - \mathbf{b}[i])^2}$$



$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Fundamentals (Distance Metrics)

Example

- The Euclidean distance between instances *d*12 (SPEED= 5.00, AGILITY= 2.5) and *d*5 (SPEED= 2.75,AGILITY= 7.5) is:

$$Euclidean(\langle 5.00, 2.50\rangle , \langle 2.75, 7.50\rangle)$$
$$= 5.4829$$

K Nearest Neighbours (K-NN) – Exercise 1

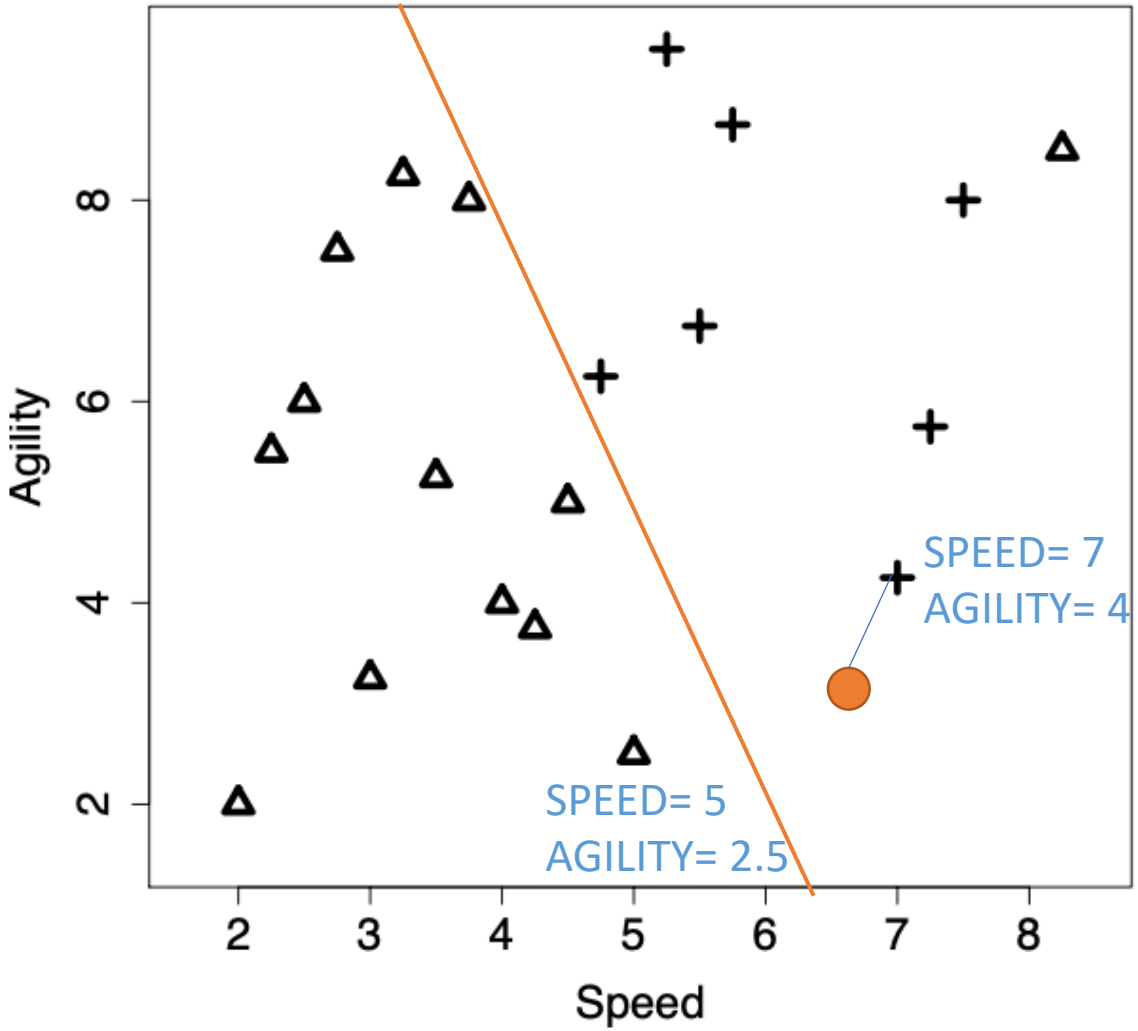
Query: SPEED= 6.75, AGILITY= 3

IS 🟠 "Draft" or "Non-draft"? **DRAFT!**

Distance to PLUS sign ("Draft") ?
Distance to Triangle sign ("Non-Draft") ?

$$Euclidean(\mathbf{a},\mathbf{b}) = \sqrt{\sum_{i=1}^{m}(\mathbf{a}[i]-\mathbf{b}[i])^2}$$

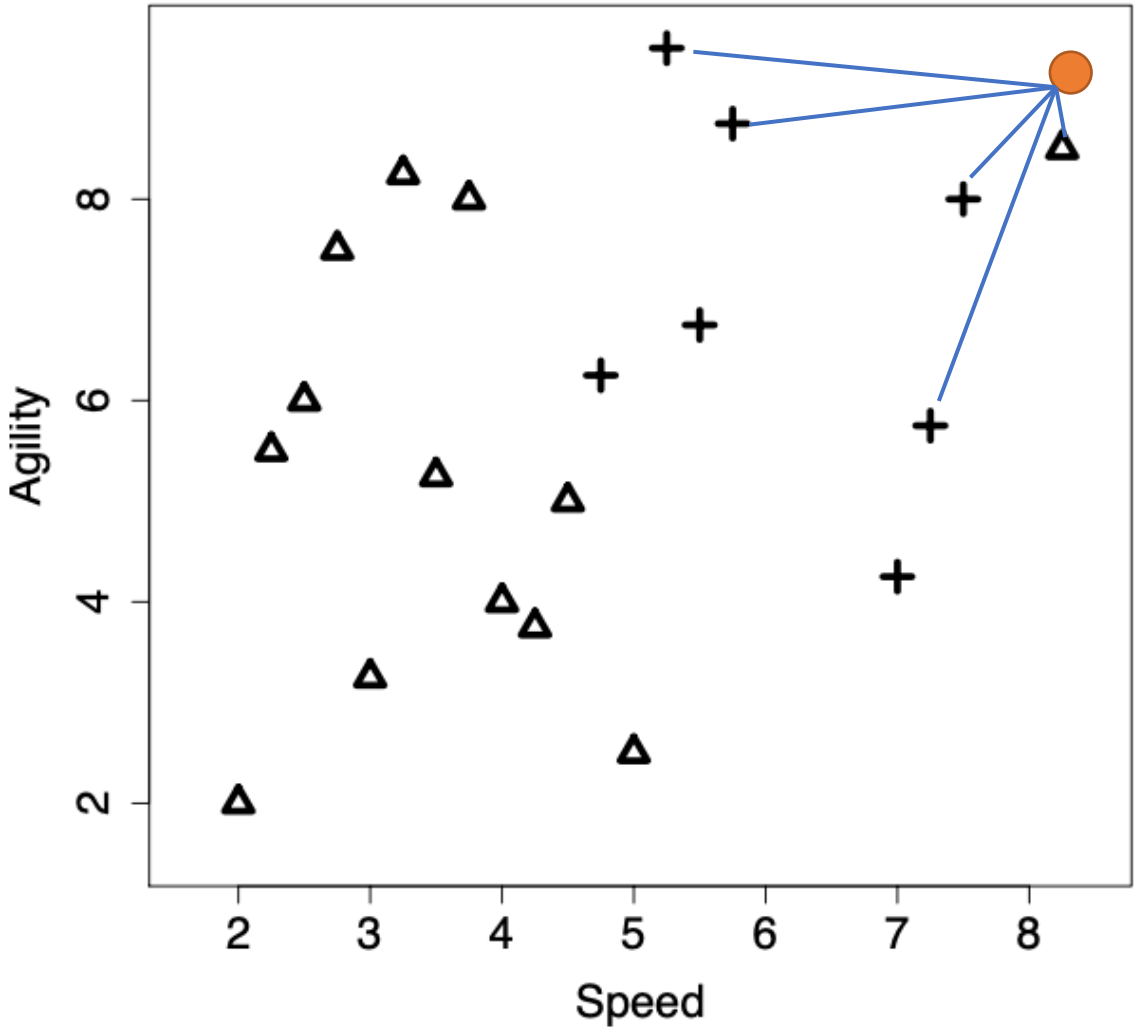
Distance to PLUS sign: root[$(7-6.75)^2 + (4-3)^2$]



K Nearest Neighbours (K-NN) – Exercise 2

Question: Should we select an athlete with the following profile this year?

Query: SPEED = 8.5, AGILITY= 9



IS ● "Draft" or "Non-draft

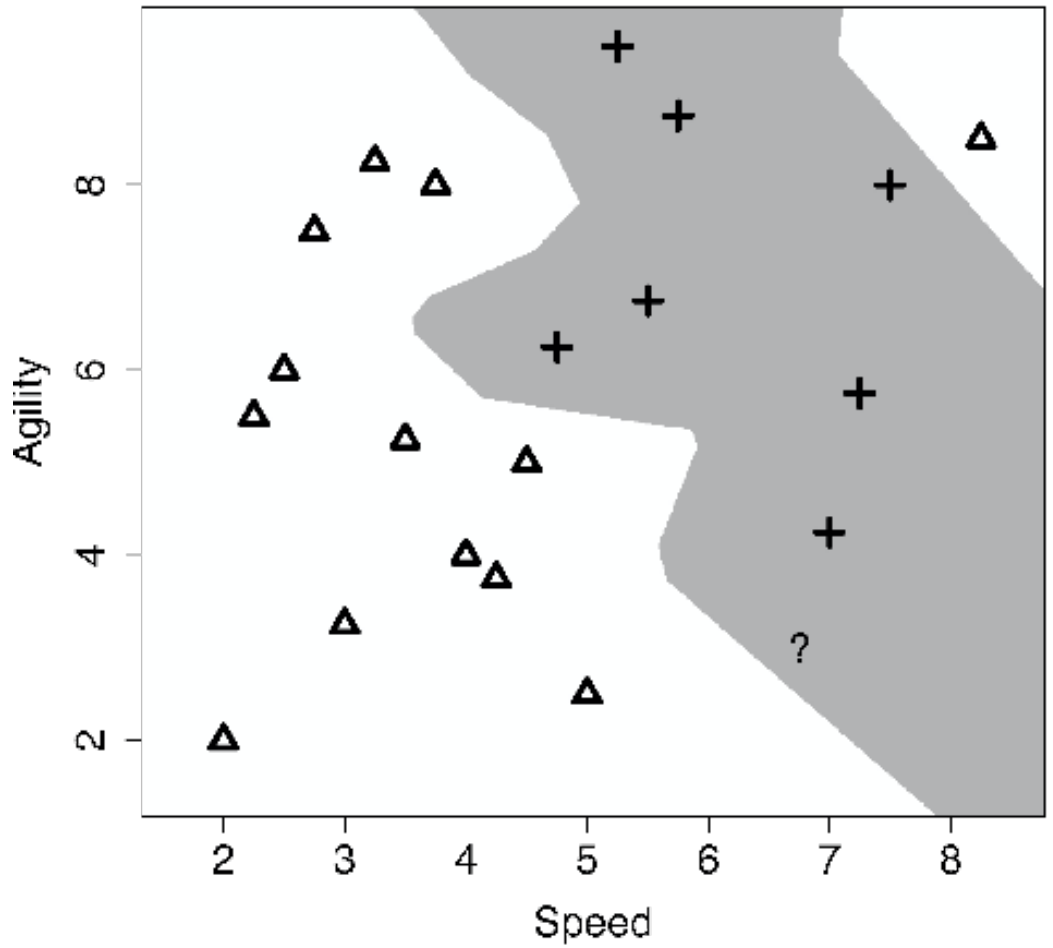
Using 1-NN? non-drafted

Using 3-NN? drafted

Using 5-NN? drafted

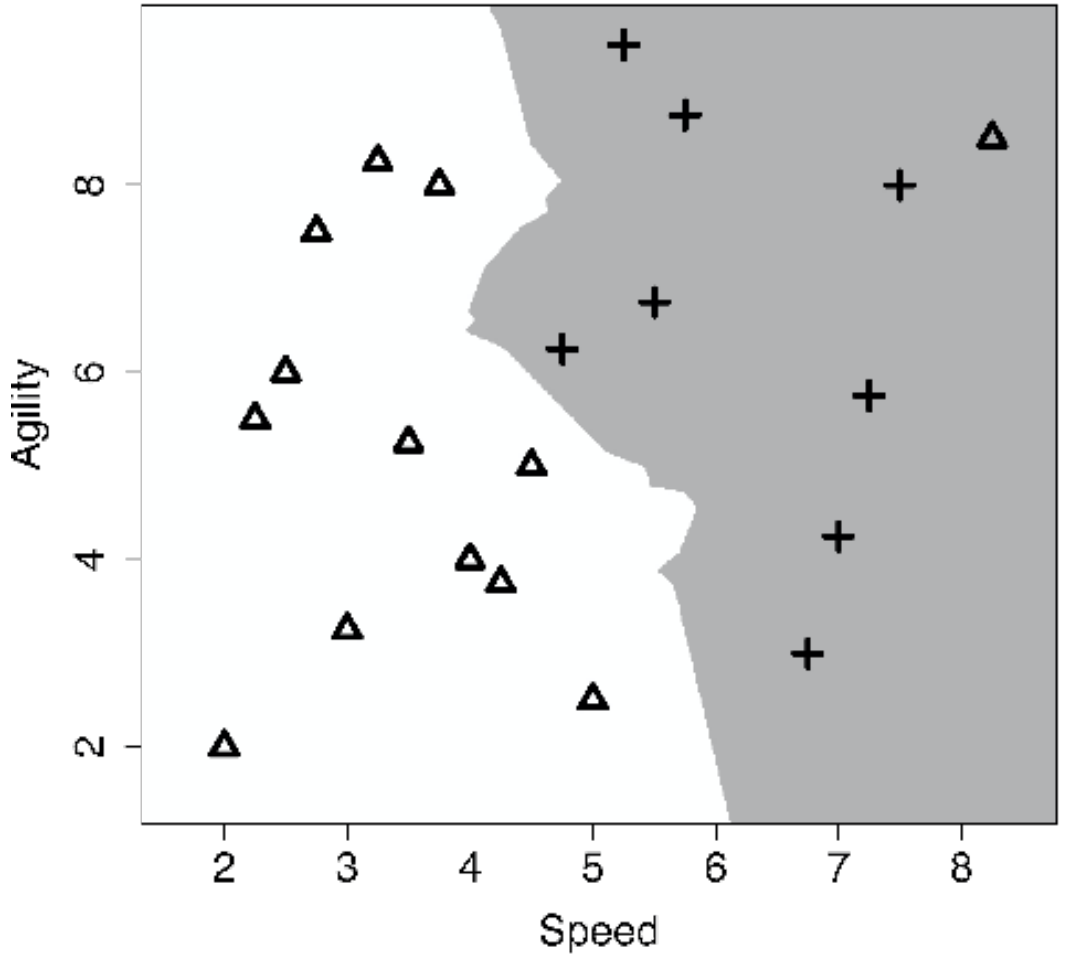
Handling Noisy Data

Figure: Is the instance at the top right of the diagram really *noise*?



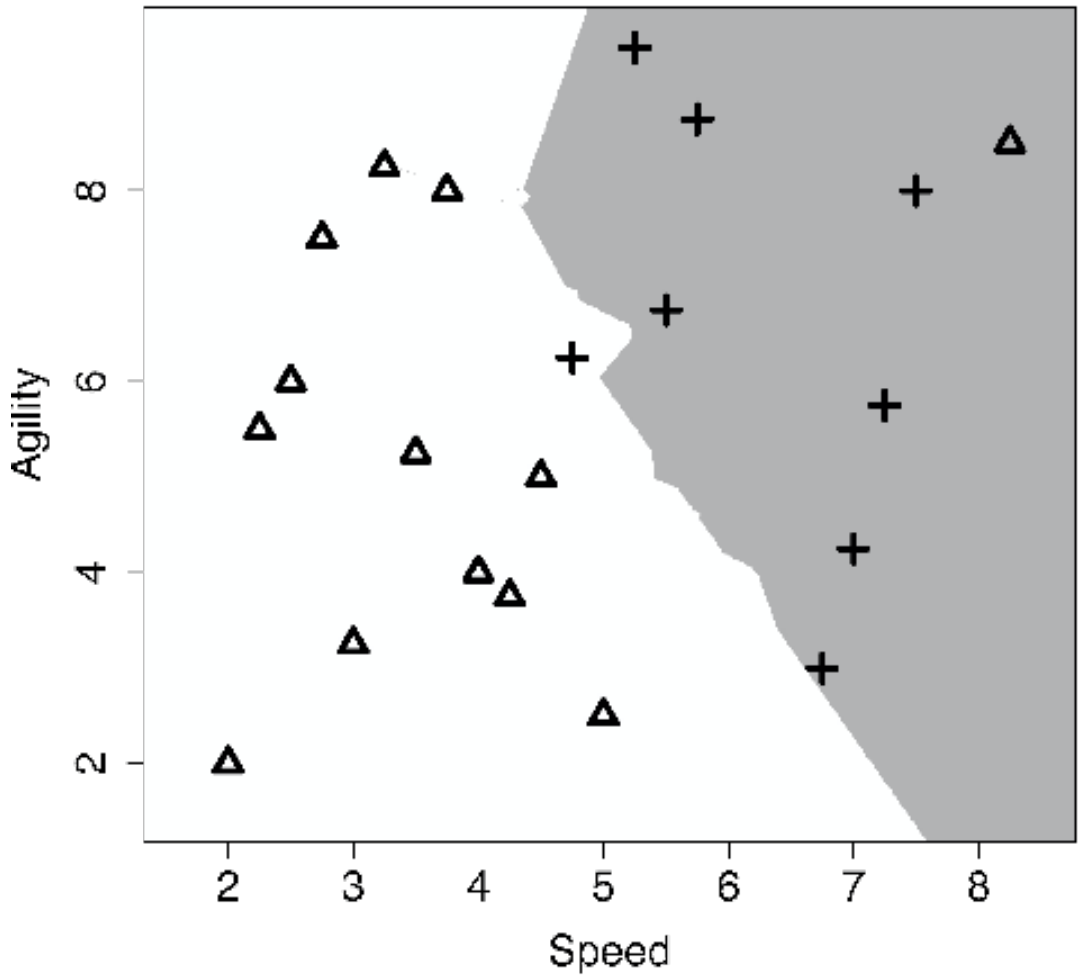
Handling Noisy Data

Figure: The decision boundary using majority classification of the nearest **3 neighbors**.

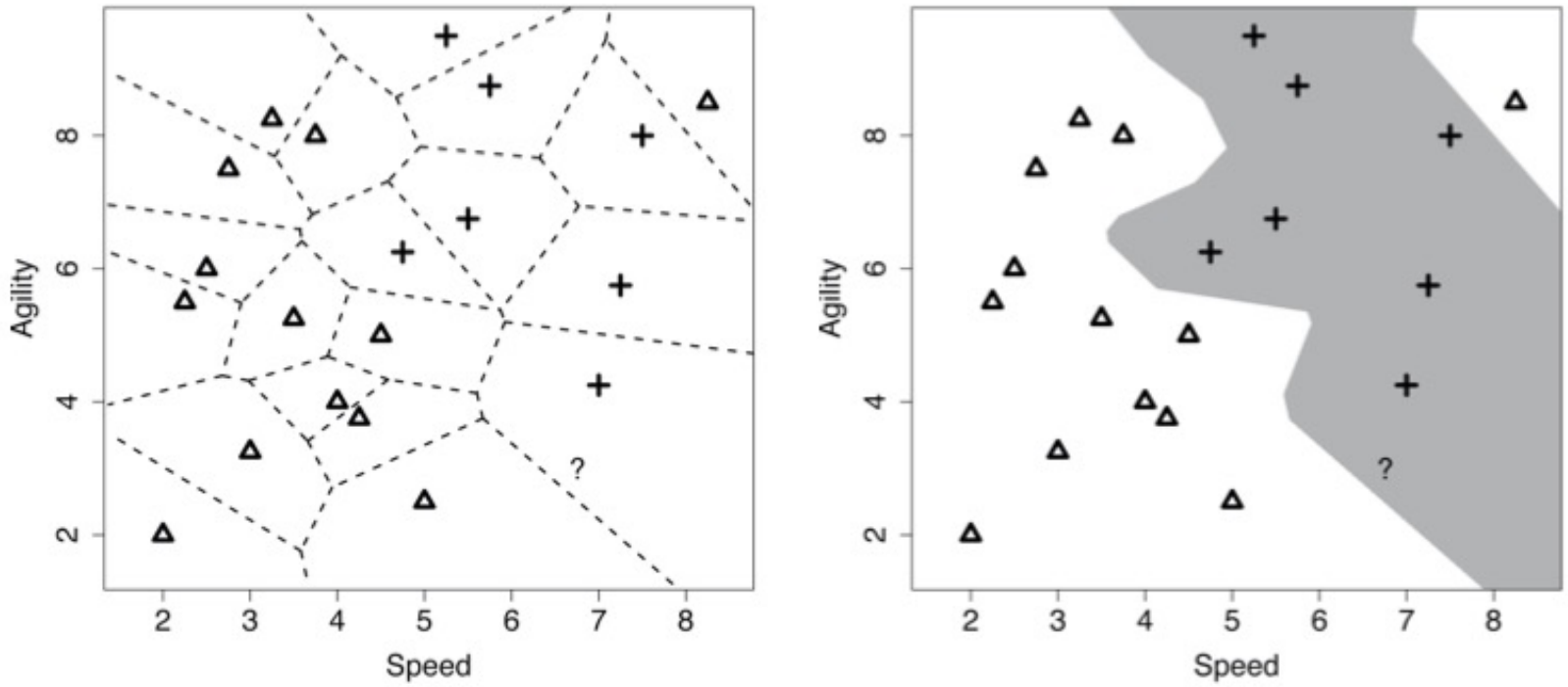


Handling Noisy Data

Figure: The decision boundary using majority classification of the nearest **5 neighbors**.



Fundamentals (Distance Metrics)



(a) Voronoi tessellation (b) Decision boundary ($k = 1$)

(a) The Voronoi tessellation of the feature space for the dataset
(b) the decision boundary created by aggregating the neighboring Voronoi regions that belong to the same target level.

Fundamentals (Distance Metrics)

One of the great things about nearest neighbour algorithms is that we can add in new data to update the model very easily.

Fundamentals ~~(Decision Trees)~~

The Nearest Neighbour Algorithm

- **Require:** set of training instances
- **Require:** a query to be classified

Instance-based learning

1: Iterate across the instances in memory and find the instance that is shortest distance from the query position in the feature space.

2: Make a prediction for the query equal to the value of the target feature of the nearest neighbor.

K- NN Algorithm (summary)

Question:

K-NN is instance-based learning and supervised

What about K-means?

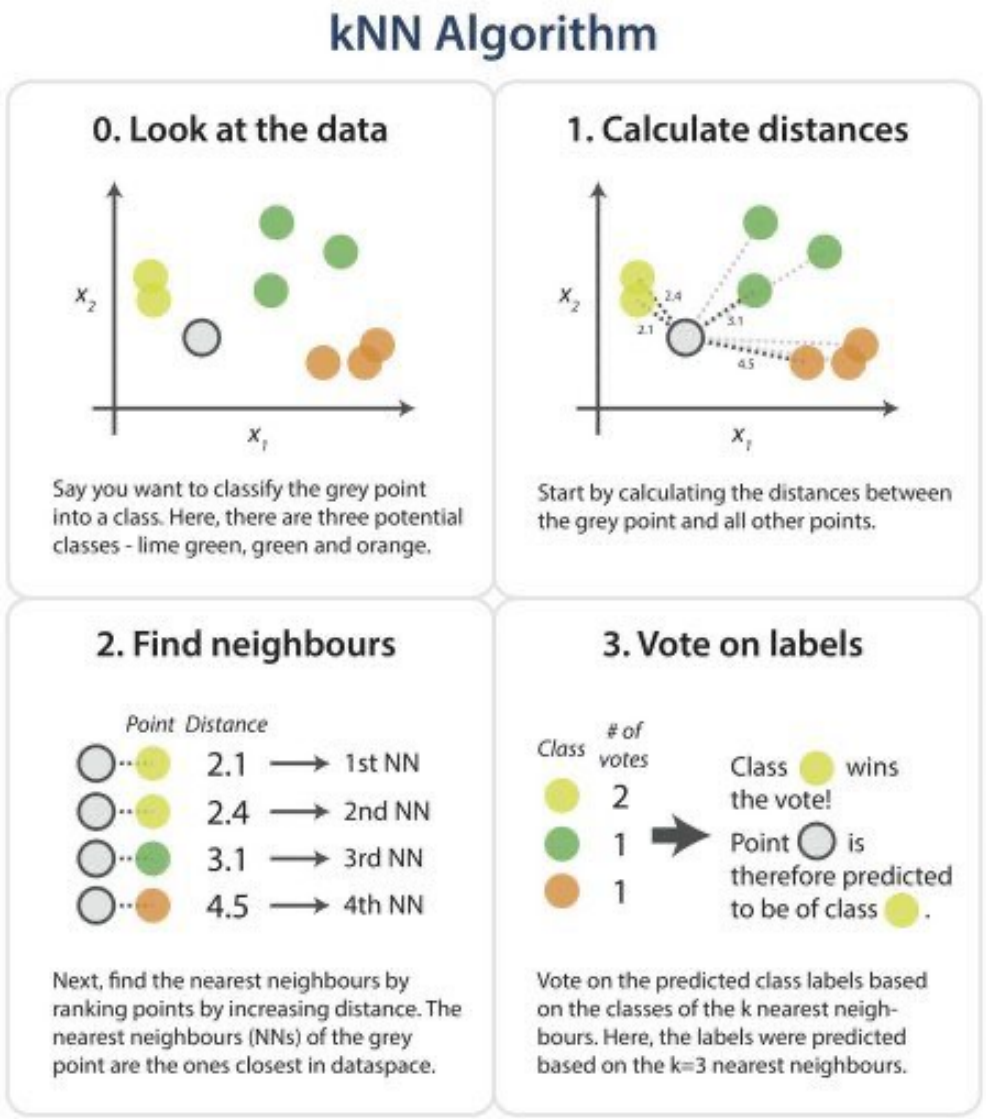
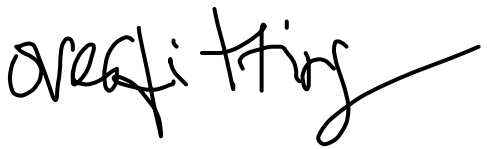


Photo via kdnuggets.com

How to tune the K in k-NN ?

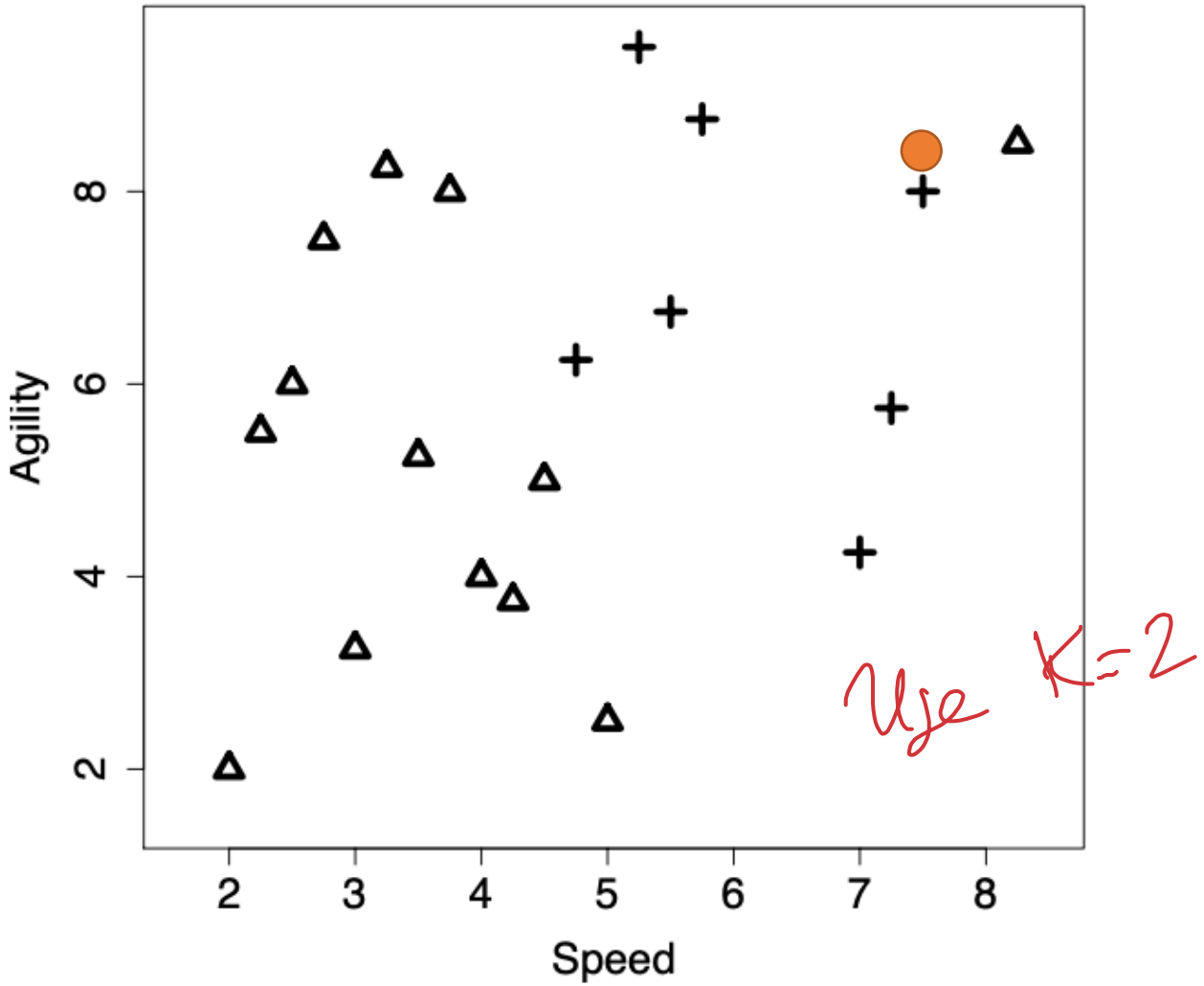
What happens if we select a very small value for K?

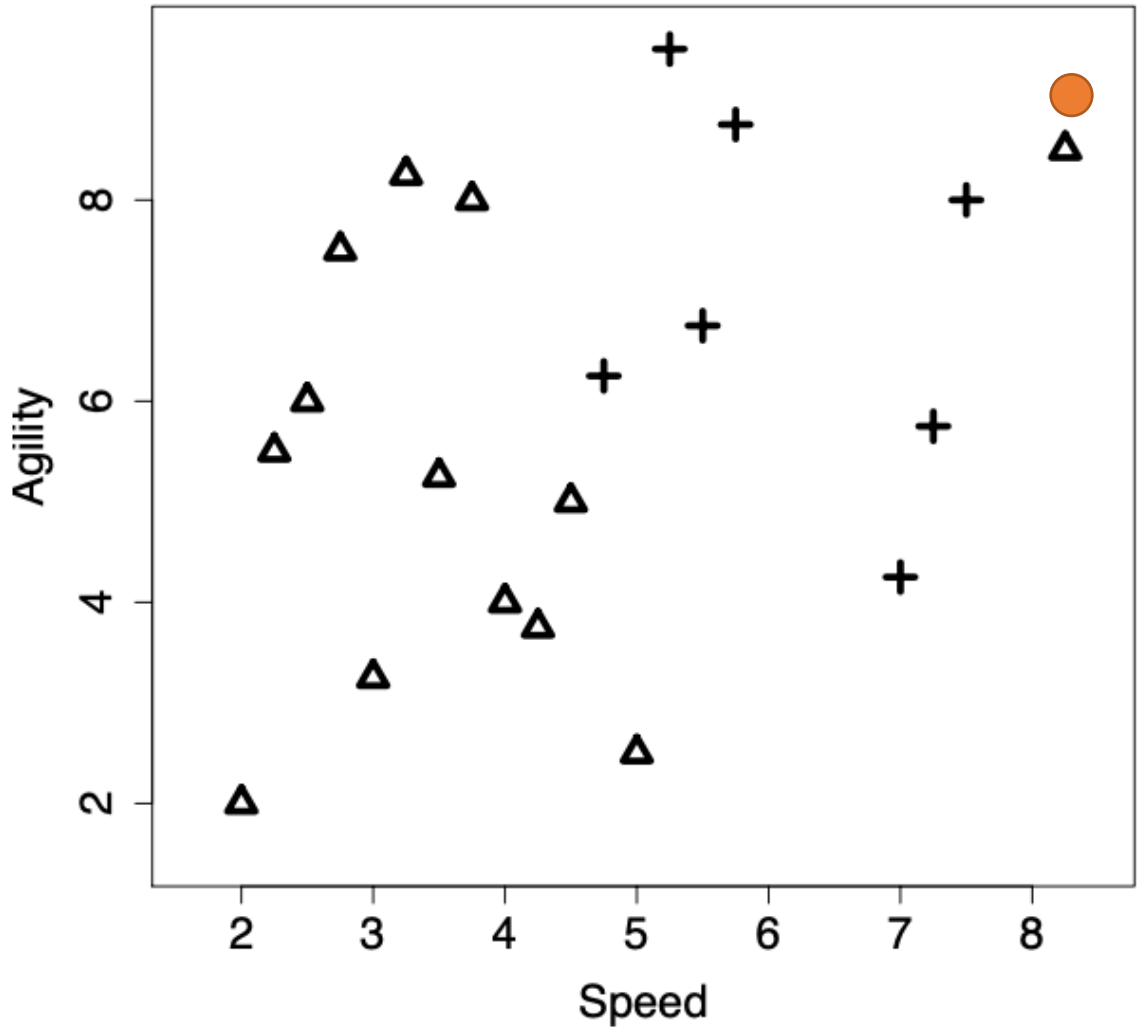


What happens if we select a very large value for K?

Dealing with a tie (draw) situation

Only possible when K is an even number Why?





Dealing with a tie (draw) situation

Only possible when K is an even number

In a distance weighted *k* nearest neighbor algorithm the contribution of each neighbor to the classification decision is weighted by the reciprocal of the squared distance between the neighbor **d** and the query **q**:

$$1 / [dist(\mathbf{q}, \mathbf{d})]$$

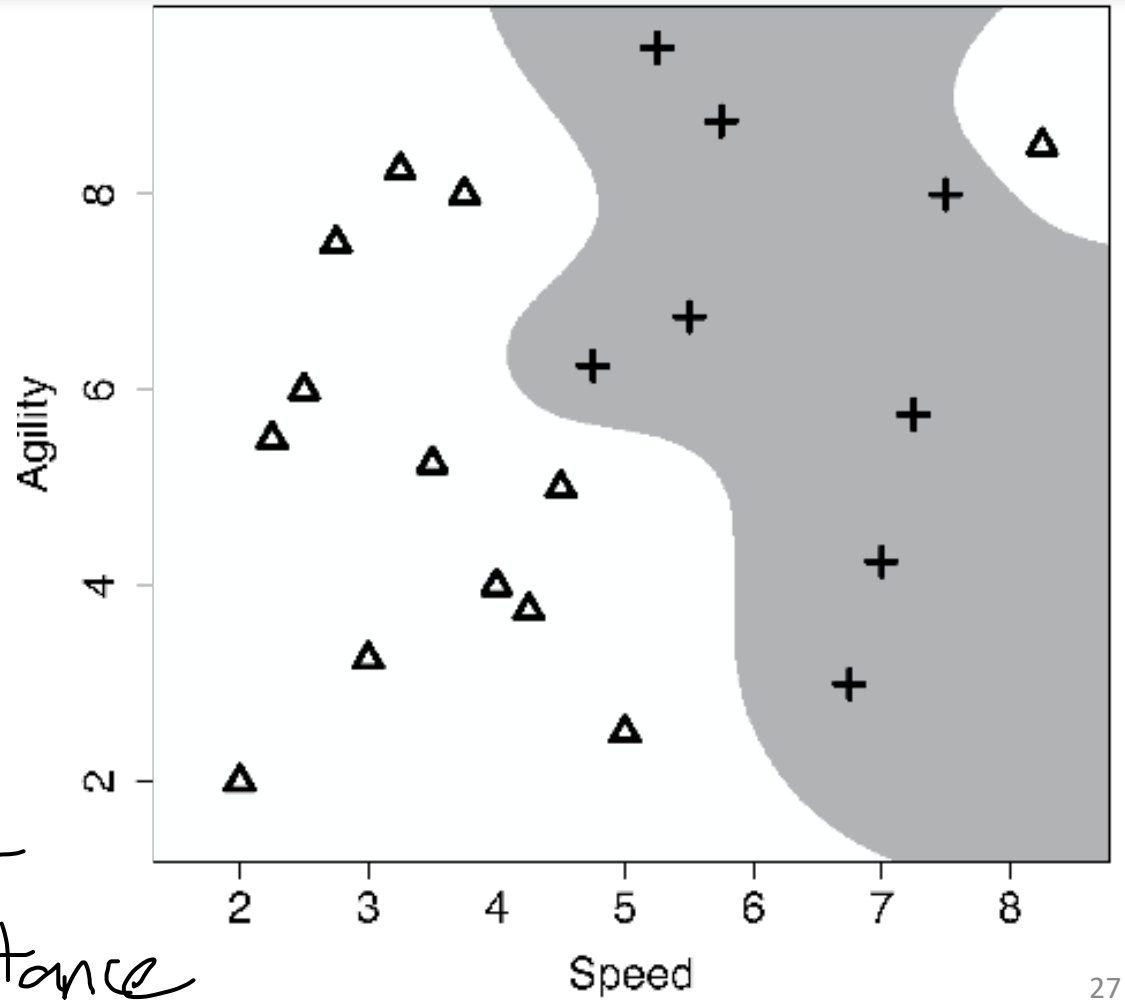
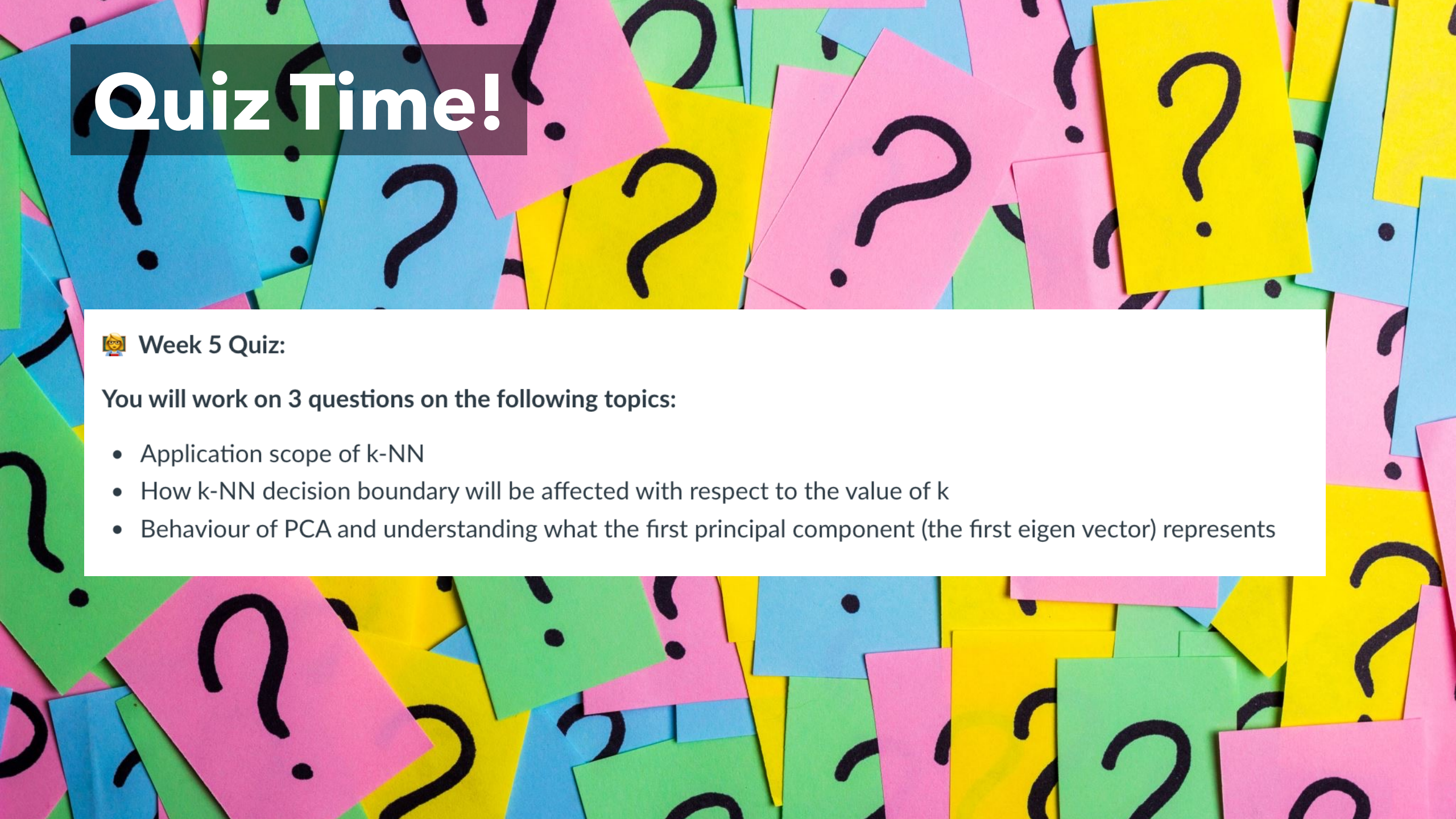


Figure: The weighted KNN decision boundary.

Voting is now weighted by the reciprocal distance

Quiz Time!



👩‍🏫 **Week 5 Quiz:**

You will work on 3 questions on the following topics:

- Application scope of k-NN
- How k-NN decision boundary will be affected with respect to the value of k
- Behaviour of PCA and understanding what the first principal component (the first eigen vector) represents

Data Normalization

Data Normalization

ID	Salary	NoT	Purchased
1	53700	41	No
2	65300	37	No
3	48900	45	Yes
4	64800	49	Yes
5	44200	30	No
6	55900	57	Yes
7	48600	26	No
8	72800	60	Yes
9	45300	34	No
10	73200	52	Yes

Figure: A dataset listing the salary and age information for customers and whether or not the purchased a pension plan.

The marketing department wants to decide whether or not they should contact a customer with the following profile:
⟨SALARY = 56, 000, NoT = 35⟩ ?

Data Normalization

$$\sqrt{(Salary_{new} - Salary)^2 + (NOT_{new} - NOT)^2}$$

$$\sqrt{(Salary_{new} - Salary)^2}$$

$$\sqrt{(NOT_{new} - NOT)^2}$$

ID	Salary	NoT	Purch.	Salary and NoT Dist.	Salary and NoT Neigh.	Salary Only Dist.	Salary Only Neigh.	NoT Only Dist.	NoT Only Neigh.
1	53700	41	No	2300.0078	2	2300	2	6	4
2	65300	37	No	9300.0002	6	9300	6	2	2
3	48900	45	Yes	7100.0070	3	7100	3	10	6
4	64800	49	Yes	8800.0111	5	8800	5	14	7
5	44200	30	No	11800.0011	8	11800	8	5	5
6	55900	57	Yes	102.3914	1	100	1	22	9
7	48600	26	No	7400.0055	4	7400	4	9	3
8	72800	60	Yes	16800.0186	9	16800	9	25	10
9	45300	34	No	10700.0000	7	10700	7	1	1
10	73200	52	Yes	17200.0084	10	17200	10	17	8

It's taking $(\sqrt{\ })^2$

Data Normalization

This odd prediction is caused by features taking different ranges of values, this is equivalent to features having different variances.

We can adjust for this using normalization; the equation for range normalization is:

$$a_i' = \frac{a_i - min(a)}{max(a) - min(a)} \times (high - low) + low$$

Usually high=1

low=0

Data Normalization

	Normalized Dataset			Salary and NoT		Salary Only		NoT Only	
ID	Salary	NoT	Purch.	Dist.	Neigh.	Dist.	Neigh.	Dist.	Neigh.
1	0.3276	0.4412	No	0.1935	1	0.0793	2	0.17647	4
2	0.7276	0.3235	No	0.3260	2	0.3207	6	0.05882	2
3	0.1621	0.5588	Yes	0.3827	5	0.2448	3	0.29412	6
4	0.7103	0.6765	Yes	0.5115	7	0.3034	5	0.41176	7
5	0.0000	0.1176	No	0.4327	6	0.4069	8	0.14706	3
6	0.4034	0.9118	Yes	0.6471	8	0.0034	1	0.64706	9
7	0.1517	0.0000	No	0.3677	3	0.2552	4	0.26471	5
8	0.9862	1.0000	Yes	0.9361	10	0.5793	9	0.73529	10
9	0.0379	0.2353	No	0.3701	4	0.3690	7	0.02941	1
10	1.0000	0.7647	Yes	0.7757	9	0.5931	10	0.50000	8

Data Normalization

Normalizing the data is an important thing to do for almost all machine learning algorithms, not just nearest neighbor!

Predicting Continuous Targets

This time, instead of majority voting, we use (weighted) average for the top K nearest neighbors:

Return the average value in the neighborhood:

$$\mathbb{M}_k(\mathbf{q}) = \frac{1}{k}\sum_{i=1}^{k} t_i$$

This allows to predict a numerical value not a category

Predicting Continuous Targets

Figure: A dataset of items listing the age (in years) and the rating (between 1 and 5, with 5 being the best) and the bottle price of each whiskey.

ID	Age	Rating	Price	ID	Age	Rating	Price
1	0	2	30.00	11	19	5	500.00
2	12	3.5	40.00	12	6	4.5	200.00
3	10	4	55.00	13	8	3.5	65.00
4	21	4.5	550.00	14	22	4	120.00
5	12	3	35.00	15	6	2	12.00
6	15	3.5	45.00	16	8	4.5	250.00
7	16	4	70.00	17	10	2	18.00
8	18	3	85.00	18	30	4.5	450.00
9	18	3.5	78.00	19	1	1	10.00
10	16	3	75.00	20	4	3	30.00

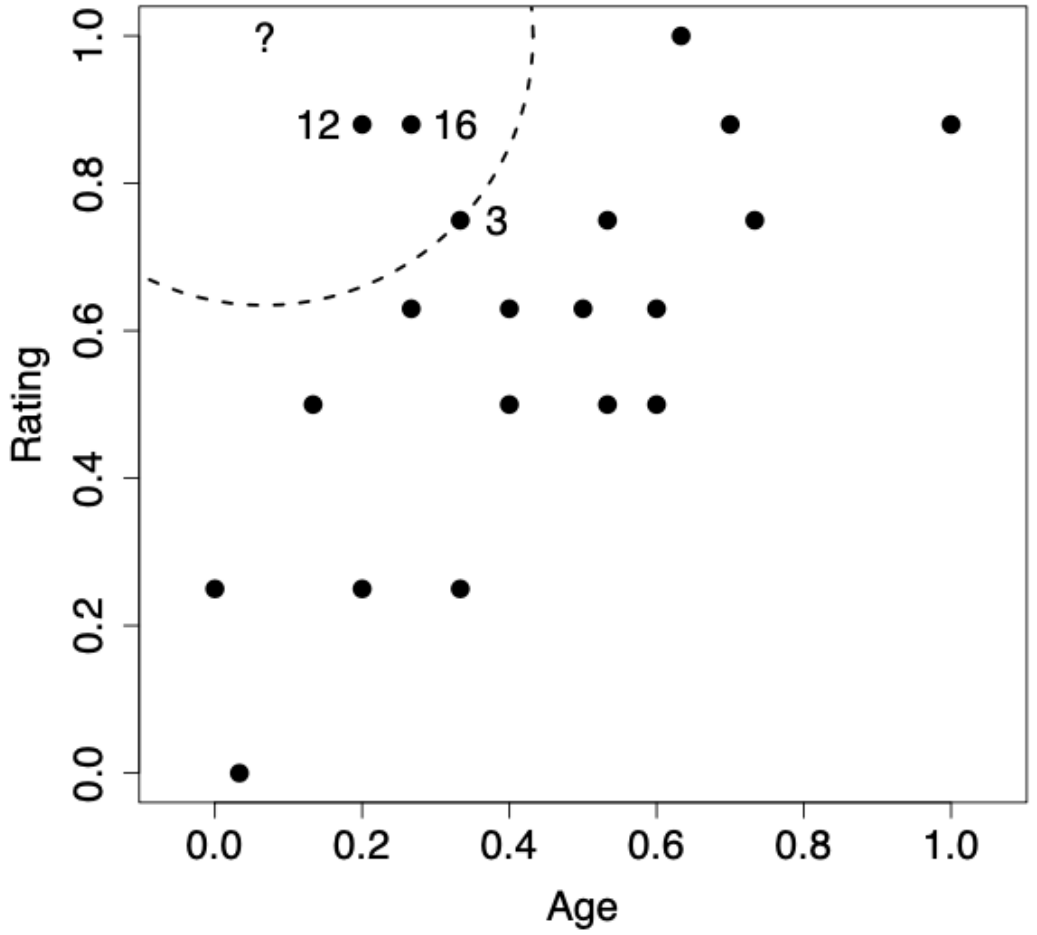
Predicting Continuous Targets

Figure: The items dataset after the descriptive features have been normalized.

ID	Age	Rating	Price	ID	Age	Rating	Price
1	0.0000	0.25	30.00	11	0.6333	1.00	500.00
2	0.4000	0.63	40.00	12	0.2000	0.88	200.00
3	0.3333	0.75	55.00	13	0.2667	0.63	65.00
4	0.7000	0.88	550.00	14	0.7333	0.75	120.00
5	0.4000	0.50	35.00	15	0.2000	0.25	12.00
6	0.5000	0.63	45.00	16	0.2667	0.88	250.00
7	0.5333	0.75	70.00	17	0.3333	0.25	18.00
8	0.6000	0.50	85.00	18	1.0000	0.88	450.00
9	0.6000	0.63	78.00	19	0.0333	0.00	10.00
10	0.5333	0.50	75.00	20	0.1333	0.50	30.00

Predicting Continuous Targets

Figure: The AGE and RATING feature space for the `items` dataset. The location of the query instance is indicated by the ? symbol. The circle plotted with a dashed line demarcates the border of the neighborhood around the query when *k* = 3. The three nearest neighbors to the query are labelled with their ID values.



Predicting Continuous Targets

The model will return a price prediction that is the average price of the three neighbors:

(200.00 + 250.00 + 55.00) / 3 = 168.33 3

Predicting Continuous Targets

Table: The calculations for the weighted *k* nearest neighbor prediction

$$\mathbb{M}_k(\mathbf{q}) = \frac{\sum_{i=1}^{k} \frac{1}{dist(\mathbf{q}, \mathbf{d}_i)^2} \times t_i}{\sum_{i=1}^{k} \frac{1}{dist(\mathbf{q}, \mathbf{d}_i)^2}}$$

ID	Price	Distance	Weight	Price × Weight
1	30.00	0.7530	1.7638	52.92
2	40.00	0.5017	3.9724	158.90
3	55.00	0.3655	7.4844	411.64
4	550.00	0.6456	2.3996	1319.78
5	35.00	0.6009	2.7692	96.92
6	45.00	0.5731	3.0450	137.03
7	70.00	0.5294	3.5679	249.75
8	85.00	0.7311	1.8711	159.04
9	78.00	0.6520	2.3526	183.50
10	75.00	0.6839	2.1378	160.33
11	500.00	0.5667	3.1142	1557.09
12	200.00	0.1828	29.9376	5987.53
13	65.00	0.4250	5.5363	359.86
14	120.00	0.7120	1.9726	236.71
15	12.00	0.7618	1.7233	20.68
16	250.00	0.2358	17.9775	4494.38
17	18.00	0.7960	1.5783	28.41
18	450.00	0.9417	1.1277	507.48
19	10.00	1.0006	0.9989	9.99
20	30.00	0.5044	3.9301	117.90
		Totals:	99.2604	16 249.85

(411.64 + 5987.53 + 4494.38) / (7.4844 + 29.9376 + 17.9775)
= ~ 197

K-NN Demos

Best Demos

- http://vision.stanford.edu/teaching/cs231n-demos/knn/
- http://sleepyheads.jp/apps/knn/knn.html

Pros and Cons of K-NN

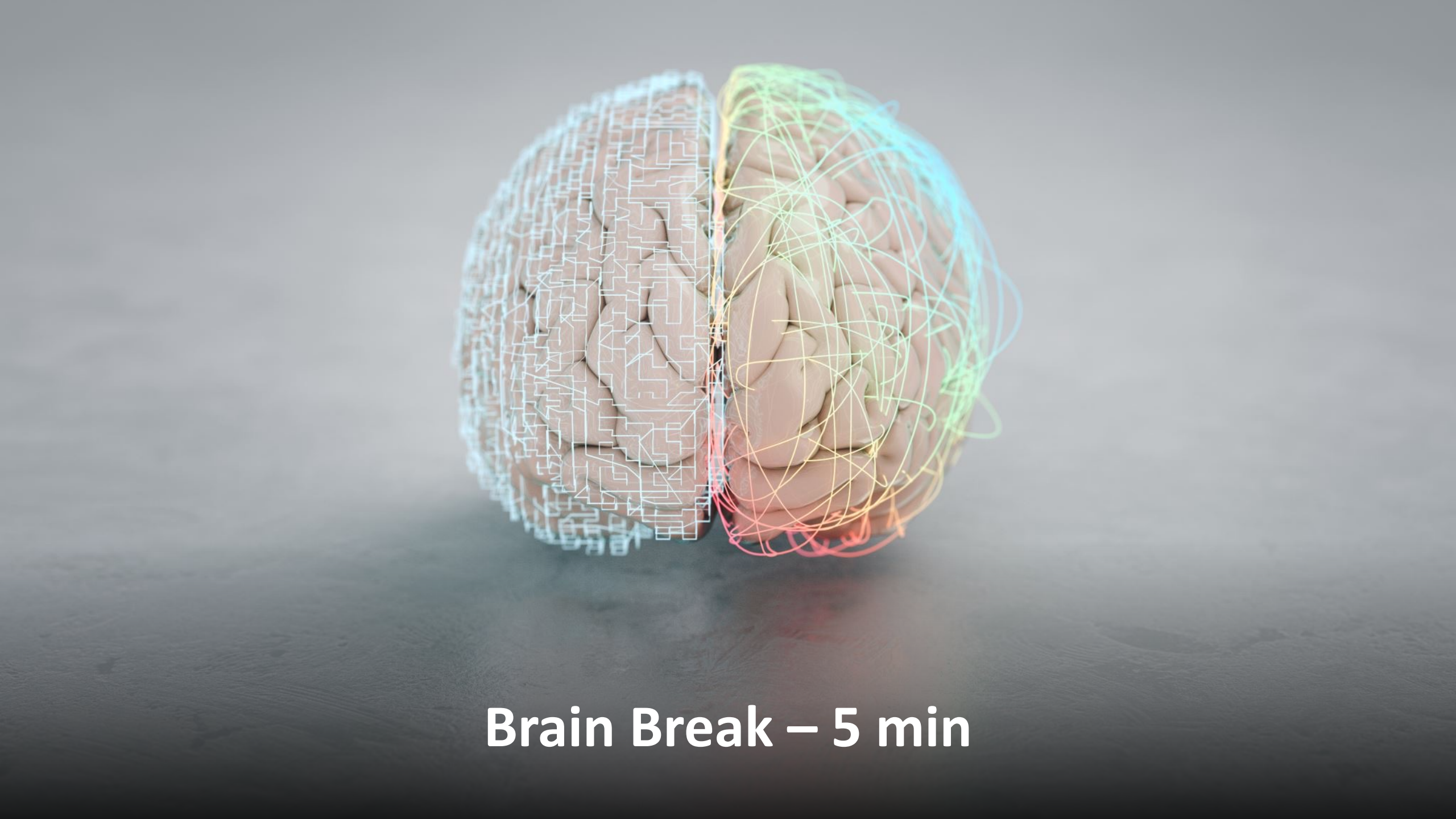
Advantages of K-NN:

1. **K-NN is pretty intuitive and simple**: K-NN algorithm is very simple to understand and equally easy to implement. To classify the new data point K-NN algorithm reads through whole dataset to find out K nearest neighbors.
2. **K-NN has no assumptions**: K-NN is a non-parametric algorithm which means there are assumptions to be met to implement K-NN. Parametric models like linear regression has lots of assumptions to be met by data before it can be implemented which is not the case with K-NN.
3. **No Training Step:** K-NN does not explicitly build any model, it simply tags the new data entry based learning from historical data. New data entry would be tagged with majority class in the nearest neighbor.
4. **It constantly evolves**: Given it's an instance-based learning; k-NN is a memory-based approach. The classifier immediately adapts as we collect new training data. It allows the algorithm to respond quickly to changes in the input during real-time use.
5. **Very easy to implement for multi-class problem:** Most of the classifier algorithms are easy to implement for binary problems and needs effort to implement for multi class whereas K-NN adjust to multi class without any extra efforts.
6. **Can be used both for Classification and Regression:** One of the biggest advantages of K-NN is that K-NN can be used both for classification and regression problems.
7. **One Hyper Parameter:** K-NN might take some time while selecting the first hyper parameter but after that rest of the parameters are aligned to it.
8. **Variety of distance criteria to be choose from:** K-NN algorithm gives user the flexibility to choose distance while building K-NN model. Euclidean Distance, Hamming Distance, Manhattan Distance, or Minkowski Distance

Pros and Cons of K-NN

Disadvantages of K-NN:

1. **K-NN slow algorithm**: K-NN might be very easy to implement but as dataset grows efficiency or speed of algorithm declines very fast.
2. **Curse of Dimensionality:** KNN works well with small number of input variables but as the numbers of variables grow K-NN algorithm struggles to predict the output of new data point.
3. **K-NN needs homogeneous features**: If you decide to build k-NN using a common distance, like Euclidean or Manhattan distances, it is completely necessary that features have the same scale, since absolute differences in features weight the same, i.e., a given distance in feature 1 must means the same for feature 2.
4. **Optimal number of neighbors**: One of the biggest issues with K-NN is to choose the optimal number of neighbors to be consider while classifying the new data entry.
5. **Imbalanced data causes problems**: k-NN doesn't perform well on imbalanced data. If we consider two classes, A and B, and the majority of the training data is labeled as A, then the model will ultimately give a lot of preference to A. This might result in getting the less common class B wrongly classified.
6. **Outlier sensitivity:** K-NN algorithm is very sensitive to outliers as it simply chose the neighbors based on distance criteria.


Brain Break – 5 min

KNN

KNN is a very simple algorithm.

- Because it is so simple, it is most appropriate for **simpler cases**
- In high dimensional data, or data with complex relationships, KNN does poorly
- It can be used for both classification **and regression**:
  - In classification, you pick the most commonly occurring category within the k nearest neighbors
  - In regression, you calculate the average of the values of the k nearest neighbors
- There are only **two** design decisions to make with KNN:
  - What number should k be? Two general rules:
    - Small k values tend to overfit, Large k values tend to underfit, so we usually want something in between
    - The most common choice is $\sqrt{n}$ for n samples
  - Which distance measure should be used?
    - The default is the L2-norm, but depending on the data/context, some might use a different norm.

KNN

Return to **knn.ipynb** to work through a KNN example for the full getting to know you data set, including a couple of ways we can assess the performance of the algorithm.

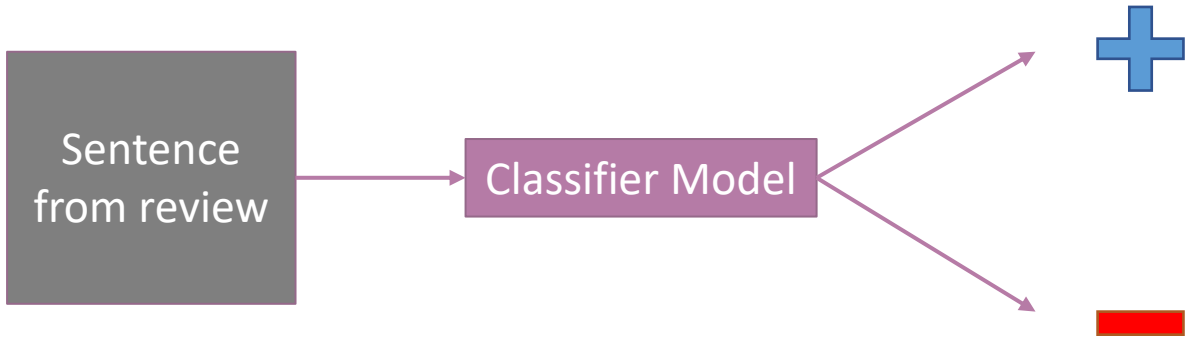
Logistic Regression Evaluation Metrics

- Generalized Linear Models
- Case Study: Sentiment Analysis
- Logistic Regression
  - Decision Boundary
  - Loss Function

- Evaluation Metrics

Sentiment Classifier

In our example, we want to classify a restaurant review as positive or negative.



Input: x

Output: y
Predicted class

Converting Text to Numbers (Vectorizing):
Bag of Words

Idea: One feature per word!

Example: ”Sushi was great, the food was awesome, but the service was terrible”

sushi	was	great	the	food	awesome	but	service	terrible

Pre-Processing: Sample Dataset

Review	Sentiment
“Sushi was great, the food was awesome, but the service was terrible”	+1
…	…
“Terrible food; the sushi was rancid.”	-1

Sushi	was	great	the	food	awesome	but	service	terrible	rancid	Sentiment
1	3	1	2	1	1	1	1	1	0	+1
…	…	…	…	…	…	…	…	…	…	…
1	1	0	1	1	0	0	0	1	1	-1

Attempt 1: Simple Threshold Classifier

Idea: Use a list of good words and bad words, classify review by the most frequent type of word

Word	Good?
sushi	None
was	None
great	Good
the	None
food	None
but	None
awesome	Good
service	None
terrible	Bad
rancid	Bad

Simple Threshold Classifier

Input x: Sentence from review

Count the number of positive and negative words, in x

If num_positive > num_negative:

- $\hat{y} = +1$

Else:

- $\hat{y} = -1$

Example: "Sushi was great, the food was awesome, but the service was terrible"

Limitations of Attempt 1 (Simple Threshold Classifier)

- Words have different degrees of sentiment.

  Awesome > Great

  How can we weigh them differently?

- Single words are not enough sometimes…

  “Good” → Positive

  “Not Good” → Negative

- How do we get list of positive/negative words?

Single Words Are Sometimes Not Enough!

What if instead of making each feature one word, we made it two?

- **Unigram**: a sequence of one word
- **Bigram**: a sequence of two words
- **N-gram**: a sequence of n-words

”Sushi was good, the food was good, the service was not good”

sushi	was	good	the	food	service	not
1	3	3	2	1	1	1

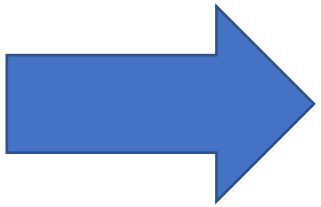
sushi was	was good	good the	the food	food was	the service	service was	was not	not good
1	2	2	1	1	1	1	1	1

Longer sequences of words results in more context, more features, and a greater chance of overfitting.

Words Have Different Degrees of Sentiments

What if we generalize good/bad to a numeric weighting per word?

Word	Good?
sushi	None
was	None
great	Good
the	None
food	None
but	None
awesome	Good
service	None
terrible	Bad
rancid	Bad



Word	Weight
sushi	0
was	0
great	1
the	0
food	0
but	0
awesome	2
service	0
terrible	-1
rancid	-2

How do we get the word weights?

What if we learn them from the data?

$$y = w0 + w_1\varphi_1(x^1) + w_2\varphi_2(x^2) + \ldots + w_D\varphi_D(x^D)$$

$\varphi_1(x)$	$\varphi_2(x)$	$\varphi_3(x)$	$\varphi_4(x)$	$\varphi_5(x)$	$\varphi_6(x)$	$\varphi_7(x)$	$\varphi_8(x)$	$\varphi_9(x)$
sushi	**was**	**great**	**the**	**food**	**awesome**	**but**	**service**	**terrible**
1	3	1	2	1	1	1	1	1

Word	**Weight**
sushi	w_1
was	w_2
great	w_3
the	w_4
food	w_5
awesome	w_6
but	w_7
service	w_8
terrible	w_9

In linear regression we learnt the weights for each feature.

Can we do something similar here?

Attempt 2: Linear Regression

$$y = w0 + w_1\varphi_1(x^1) + w_2\varphi_2(x^2) + \ldots + w_D\varphi_D(x^D)$$

Idea: Use the regression model we learnt! The output will be the sentiment!

$$\text{Predicted Sentiment} = \hat{y} = \sum_{j=0}^{D} w_j \varphi_j(x^{(i)})$$

$\varphi_1(x)$	$\varphi_2(x)$	$\varphi_3(x)$	$\varphi_4(x)$	$\varphi_5(x)$	$\varphi_6(x)$	$\varphi_7(x)$	$\varphi_8(x)$	$\varphi_9(x)$
sushi	**was**	**great**	**the**	**food**	**awesome**	**but**	**service**	**terrible**
1	3	1	2	1	1	1	1	1

Word	**Weight**
sushi	0
was	0
great	1
the	0
food	0
awesome	2
but	0
service	0
terrible	−1

”Sushi was great, the food was awesome, but the service was terrible”

$\text{Predicted Sentiment} = \hat{y}$

$= (1*1) + (2*1) + (-1*1)$

$= 2$

Attempt 2: Linear Regression

$$Score\left(x^{(i)}\right) = \hat{s}$$
$$= w0 + w_1\varphi_1(x^1) + w_2\varphi_2(x^2) + \ldots + wD\varphi_D(x^D)$$
$$= \sum_{j=0}^{D} w_j\varphi_j(x^{(i)})$$
$$= w^T\varphi(x^{(i)})$$

This score will be always numerical!

Attempt 3: Linear Classifier

Idea: Only predict the sign of the output!

$$\text{Predicted Sentiment} = \hat{y} = sign(\, Score(x))$$

Linear Classifier

Input x: Sentence from review

Compute $Score(x)$

If $Score(x) > 0$:

$\hat{y} = +1$

Else:

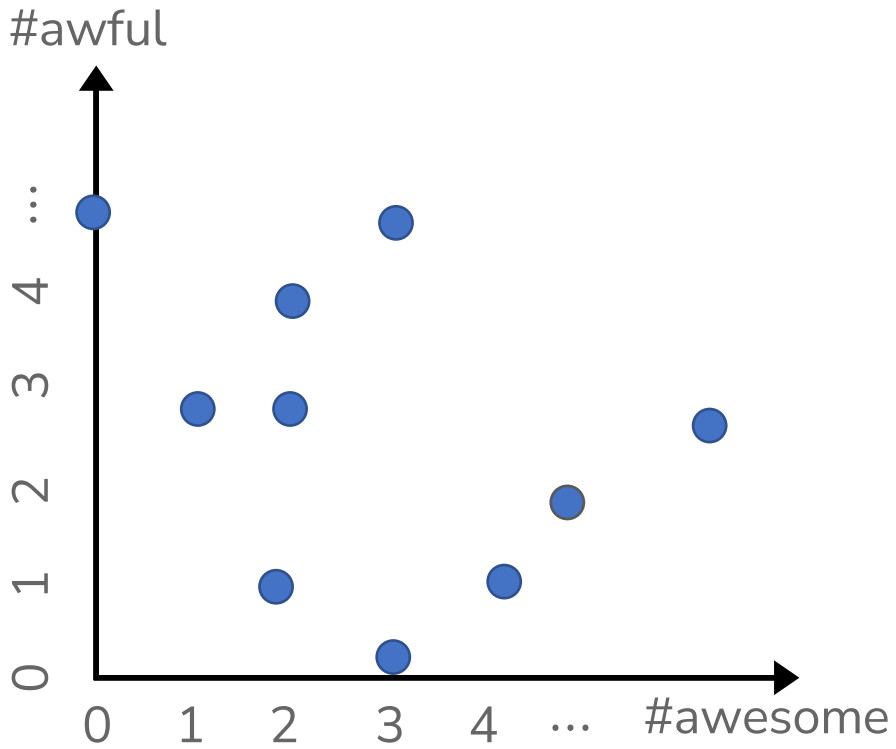
$\hat{y} = -1$

Decision Boundary

Consider if only two words had non-zero coefficients

Word	Coefficient	Weight
	w_0	0.0
awesome	w_1	1.0
awful	w_2	-1.5

$$\hat{s} = 1 \cdot \#awesome - 1.5 \cdot \#awful$$

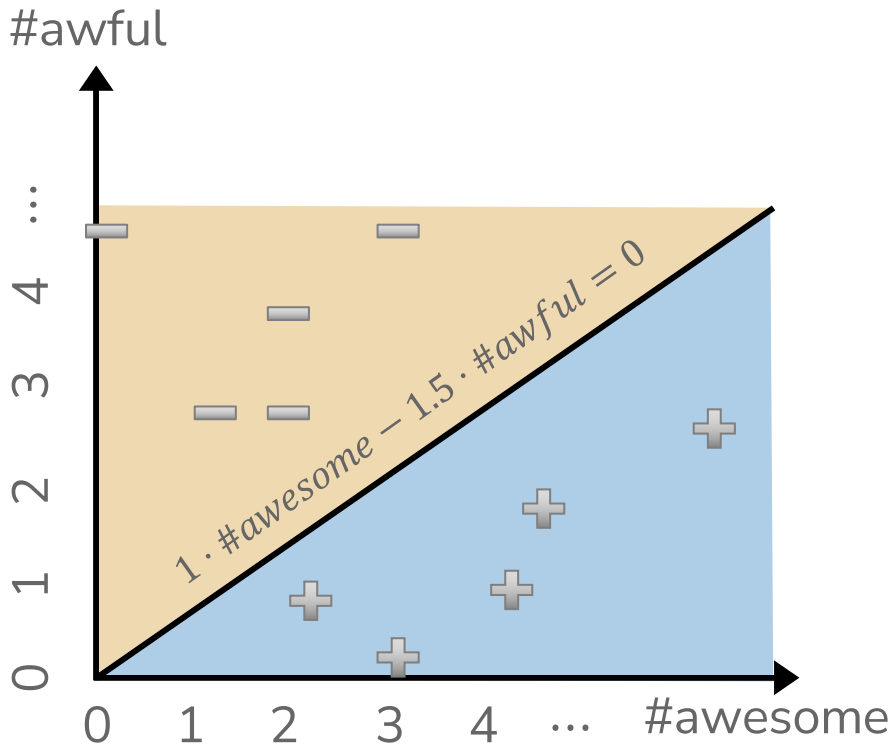


Decision Boundary

Consider if only two words had non-zero coefficients

Word	Coefficient	Weight
	w_0	0.0
awesome	w_1	1.0
awful	w_2	-1.5

$$\hat{s} = 1 \cdot \#awesome - 1.5 \cdot \#awful$$



Issue: How do we train this?

Say we were to use the MSE…

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - sign(\,Score\left(x^{(i)}\right)\,)^2$$

The derivative of the $sign$ function is 0!

Hence, Gradient Descent will no longer work ☹

Probabilities

Assume that there is some randomness in the world, and instead will try to model the probability of a positive/negative label.

Examples:

"The sushi & everything else were awesome!"

- Definite positive (+1)
- $P(y = +1 \mid x = \text{“The sushi \& everything else were awesome!”}) = 0.99$

"The sushi was alright, the service was OK"

- Not as sure
- $P(y = -1 \mid x = \text{“The sushi alright, the service was okay!”}) = 0.5$

Use probability as the measurement of certainty

$$P(y|x)$$

Probability Classifier

Idea: Estimate probabilities $\hat{P}(y|x)$ and use those for prediction

Probability Classifier

Input x: Sentence from review

Estimate class probability $\hat{P}(y = +1|x)$

If $\hat{P}(y = +1|x) > 0.5$:

- $\hat{y} = +1$

Else:

- $\hat{y} = -1$

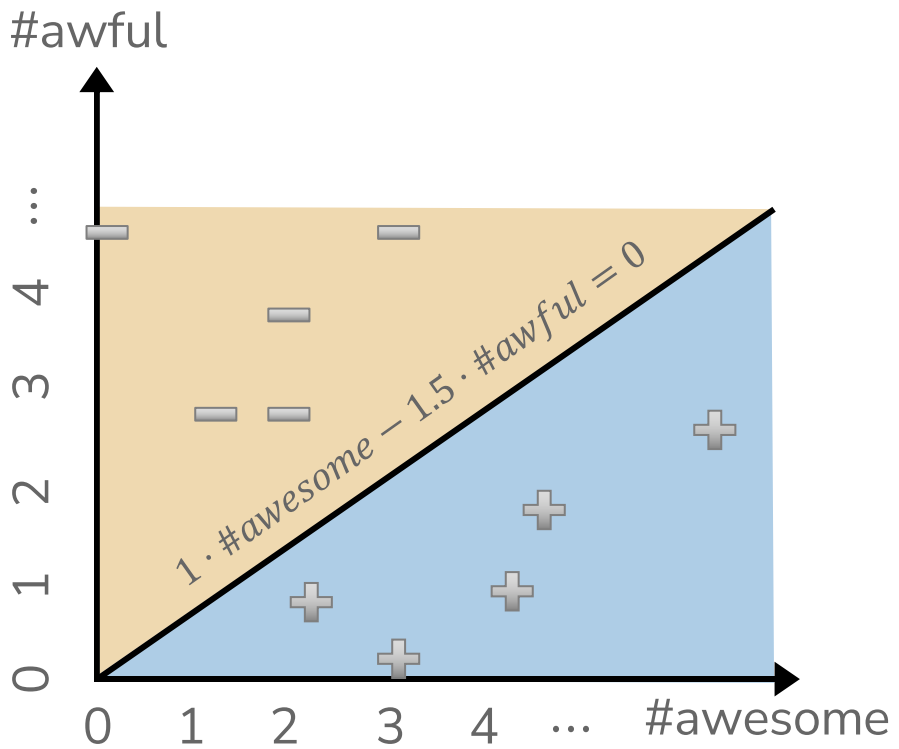
Notes:

Estimating the probability improves **interpretability.**

- Unclear how much better a score of 5 is from a score of 3. Clear how much better a probability of 0.75 is than a probability of 0.5

Connecting Score & Probability

Idea: Let's try to relate the value of $Score(x)$ to $\hat{P}(y = +1|x)$

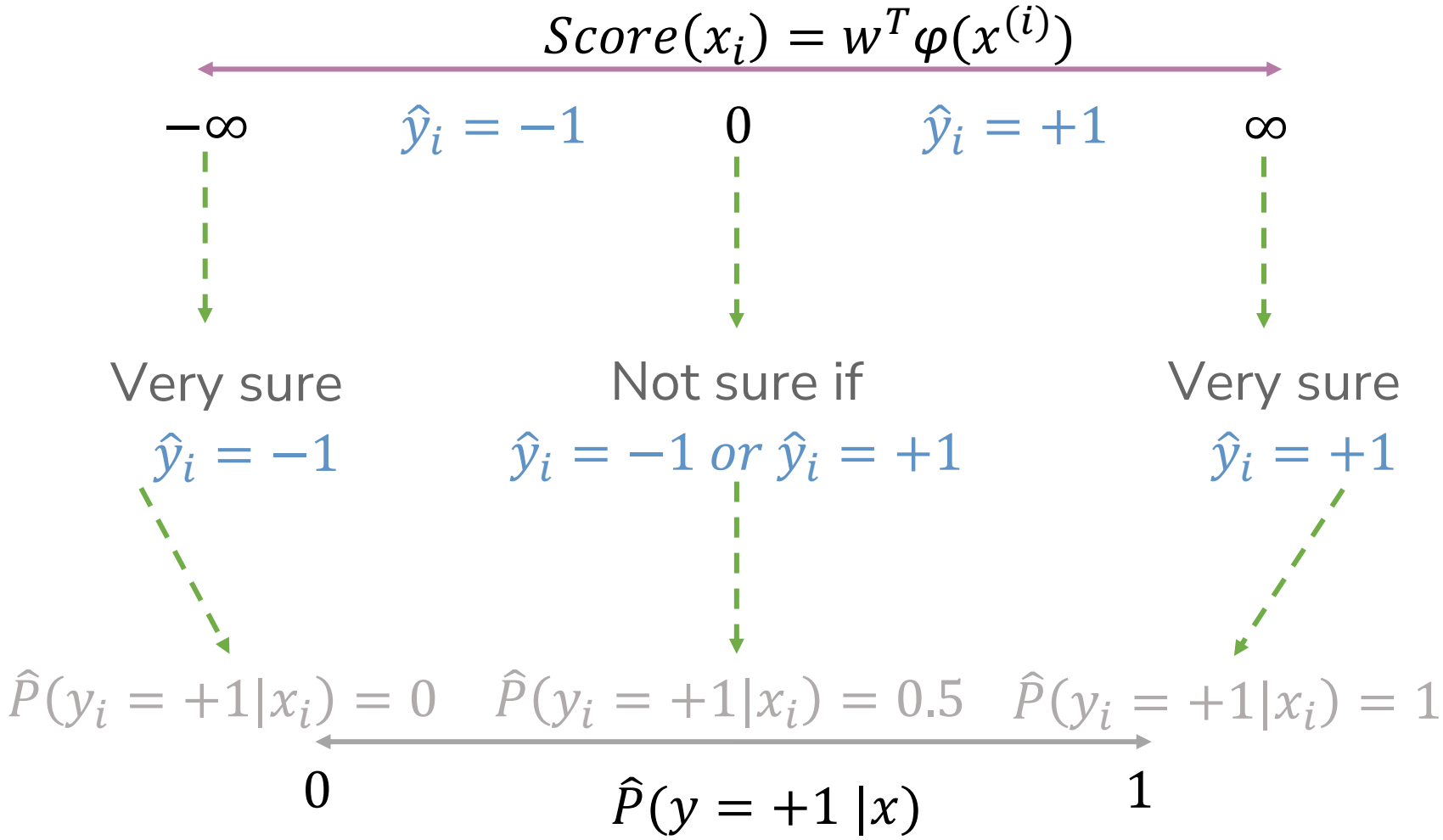


What if $Score(x)$ is positive?

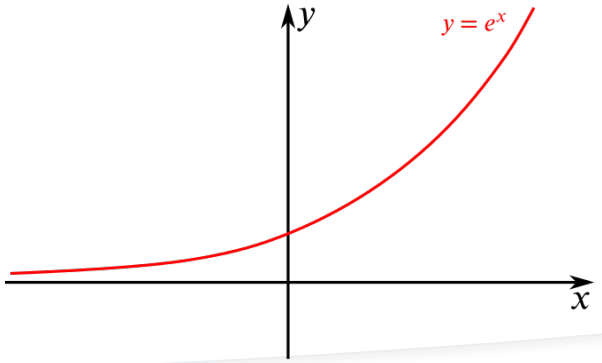
What if $Score(x)$ is negative?

What if $Score(x)$ is 0?

Connecting Score & Probability



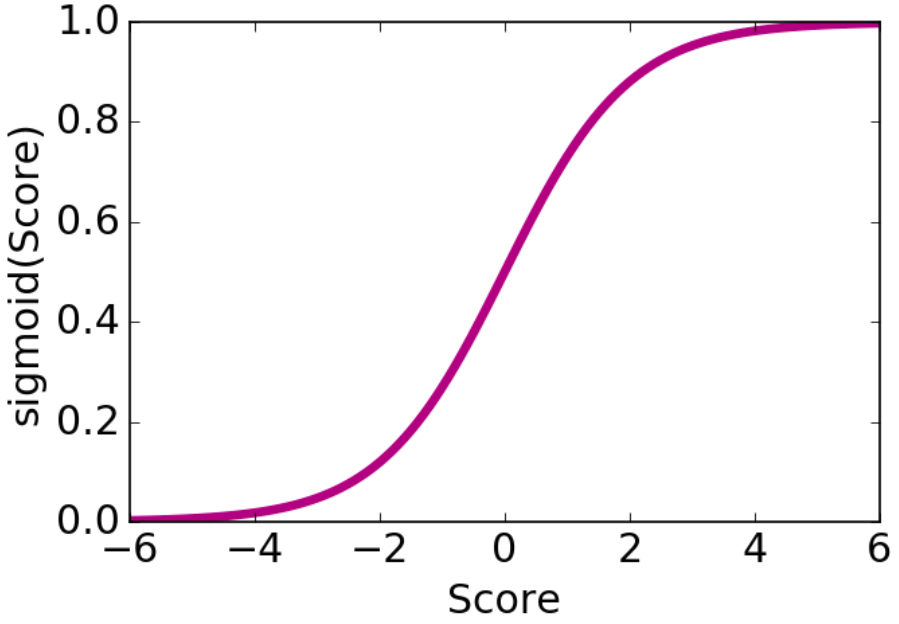
Logistic Function



Want: a function that takes numbers arbitrarily large/small and maps them between 0 and 1.

$$sigmoid(Score(x)) = \frac{1}{1 + e^{-Score(x)}}$$

$Score(x)$	$sigmoid(Score(x))$
$-\infty$	$\frac{1}{1+e^{-(-\infty)}} = 0$
-2	
0	$\frac{1}{1+e^{-(0)}} = 0.5$
2	
∞	$\frac{1}{1+e^{-(\infty)}} = 1$



Logistic Function

$$P(y_i = +1|x_i, w) = sigmoid\big(Score(x_i)\big) = \frac{1}{1 + e^{-w^T \varphi(x^{(i)})}}$$

Logistic Regression Classifier

Input x: Sentence from review

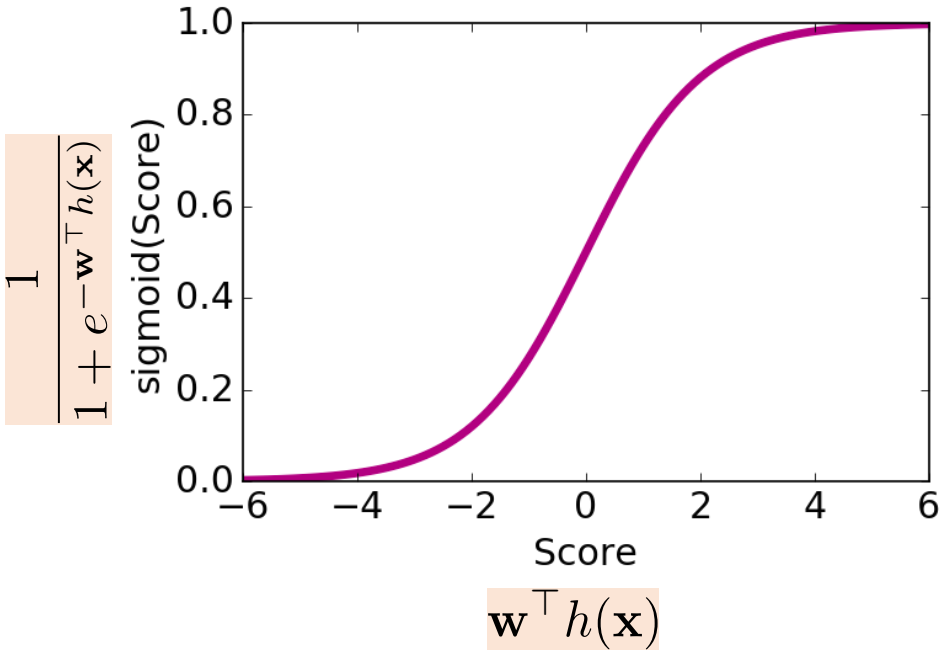
Estimate class probability $\hat{P}(y = +1|x, \hat{w}) = sigmoid(\hat{w}^T h(x_i))$

If $\hat{P}(y = +1|x, \hat{w}) > 0.5$:

$$\hat{y} = +1$$

Else:

$$\hat{y} = -1$$



Quality Metric = Likelihood

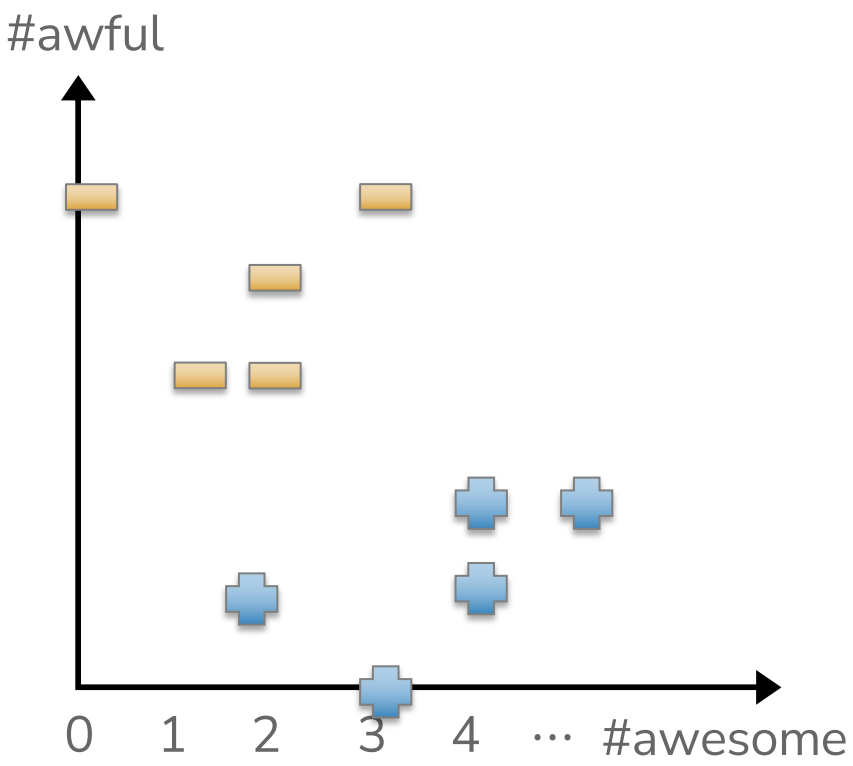
Want to compute the probability of seeing our dataset for every possible setting for w. Find w that makes data most likely!

Data Point	$\varphi_1(x)$	$\varphi_2(x)$	y	Choose w to maximize
$x^{(1)}, y^{(1)}$	2	1	$+1$	$P\left(y^{(1)} = +1 \middle\| x^{(1)}, w\right)$
$x^{(2)}, y^{(2)}$	0	2	-1	$P\left(y^{(2)} = -1 \middle\| x^{(2)}, w\right)$
$x^{(3)}, y^{(3)}$	3	3	-1	$P\left(y^{(3)} = -1 \middle\| x^{(3)}, w\right)$
$x^{(4)}, y^{(4)}$	4	1	$+1$	$P\left(y^{(4)} = +1 \middle\| x^{(4)}, w\right)$

Learn $\widehat{w}$

Now that we have our new model, we will talk about how to choose $\widehat{w}$ to be the "best fit".

The choice of w affects how likely seeing our dataset is



$$\ell(w) = \prod_{i}^{n} P(y^{(i)}|x^{(i)}, w)$$

$$P\left(y^{(i)} = +1 \middle| x^{(i)}, w\right) = \frac{1}{1 + e^{-w^T h(x^{(i)})}}$$

$$P\left(y^{(i)} = -1 \middle| x^{(i)}, w\right) = \frac{e^{-w^T h(x^{(i)})}}{1 + e^{-w^T h(x^{(i)})}}$$

Loss Function

Find the w that maximizes the likelihood

$$\hat{w} = \underset{w}{\operatorname{argmax}}\, \ell(w) = \underset{w}{\operatorname{argmax}} \prod_{i=1}^{n} P(y_i|x_i, w)$$

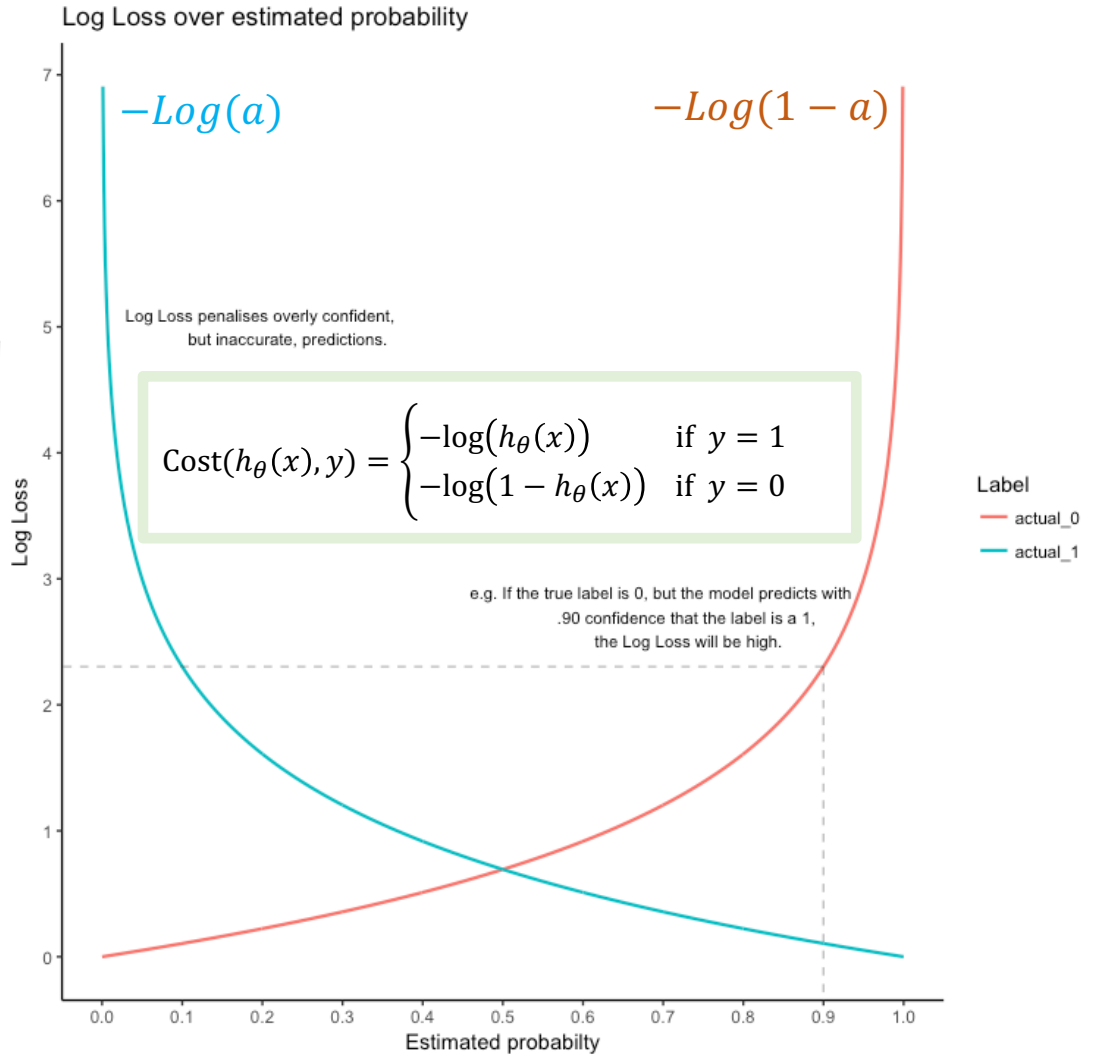
Generally, we maximize the log-likelihood which looks like

$$\hat{w} = \underset{w}{\operatorname{argmax}}\, \ell(w) = \underset{w}{\operatorname{argmax}} \log(\ell(w)) = \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} \log(P(y_i|x_i, w))$$

Also commonly written by separating out positive/negative terms

$$LogLoss = -\frac{1}{n}\sum_{i=0}^{n}[y_i log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

For Positive terms *For Negative terms*



Decision Boundary

The decision boundary is the set of x such that

$$\frac{1}{1 + e^{-(w^T X)}} = 0.5$$

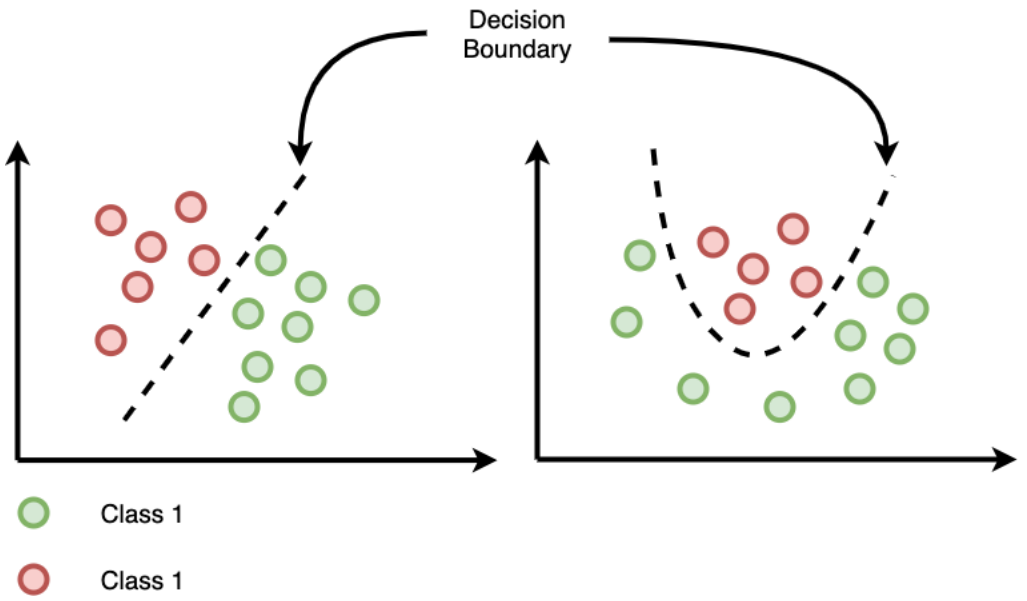
A little bit of algebra shows that this is equivalent to

$$1 = e^{-(w^T X)}$$

and, taking the natural log of both sides,

$$0 = -\sum_{i=0}^{D} w_i x_i$$

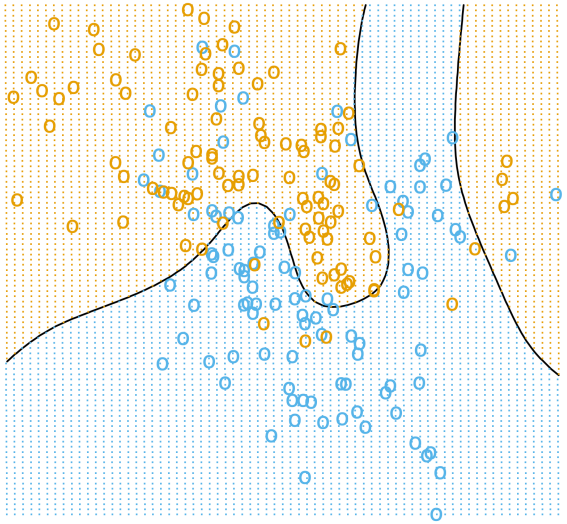
So, our decision boundary is linear!

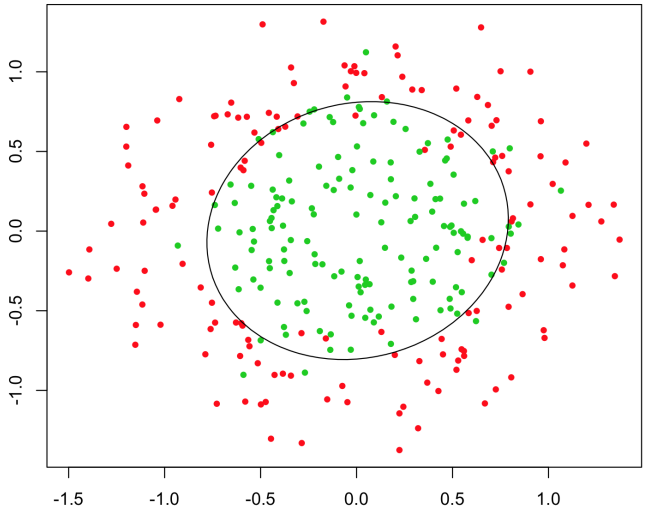


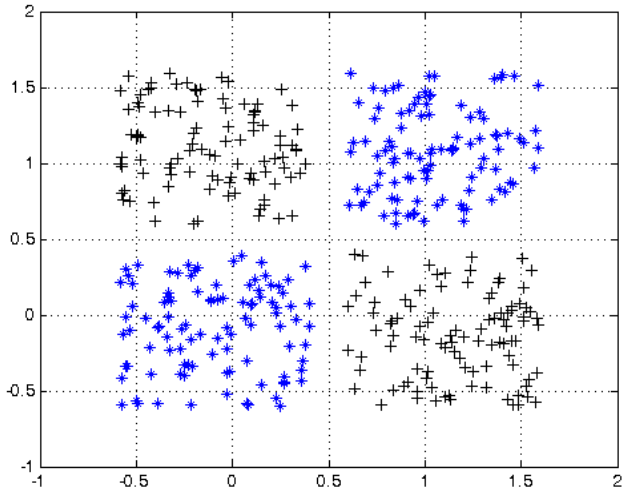
Complex Decision Boundaries?

What if we want to use a more complex decision boundary?

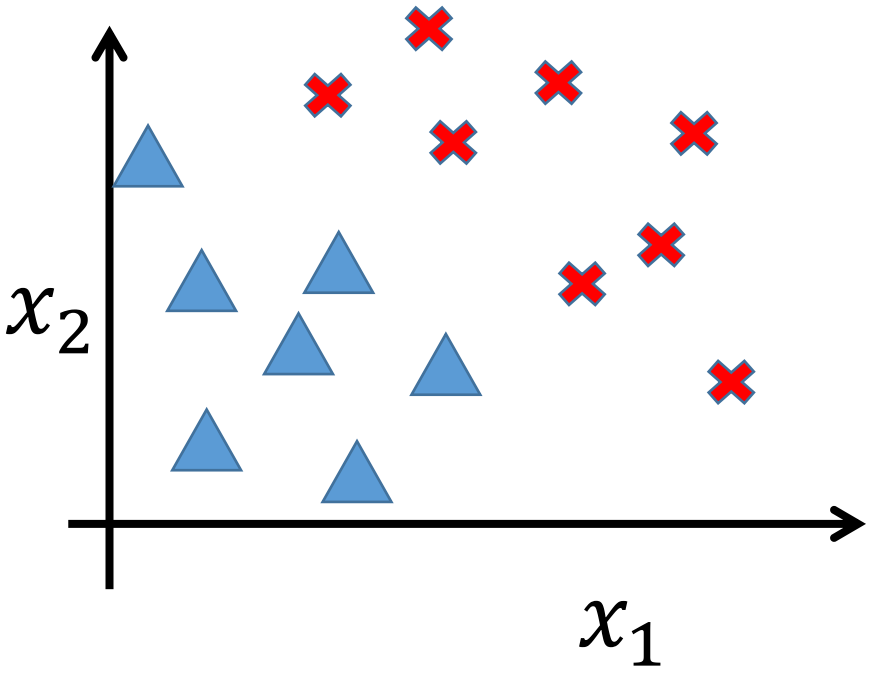
- Need more complex model/features!



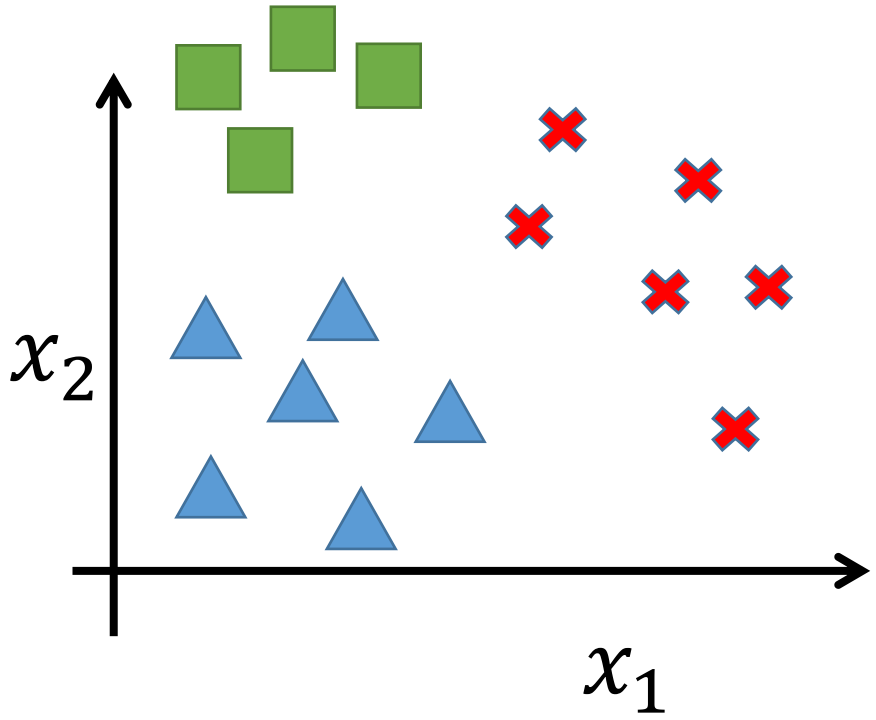




Binary classification

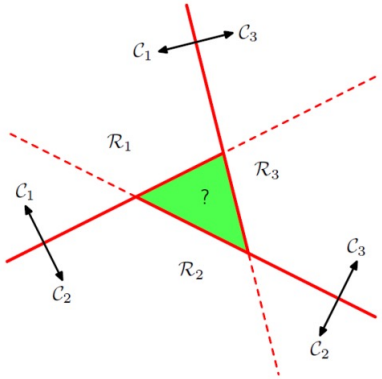


Multiclass classification



How do we extend Logistic Regression to Multiclass classification?

- Approach 1: one-versus-one
  - Computationally very expensive

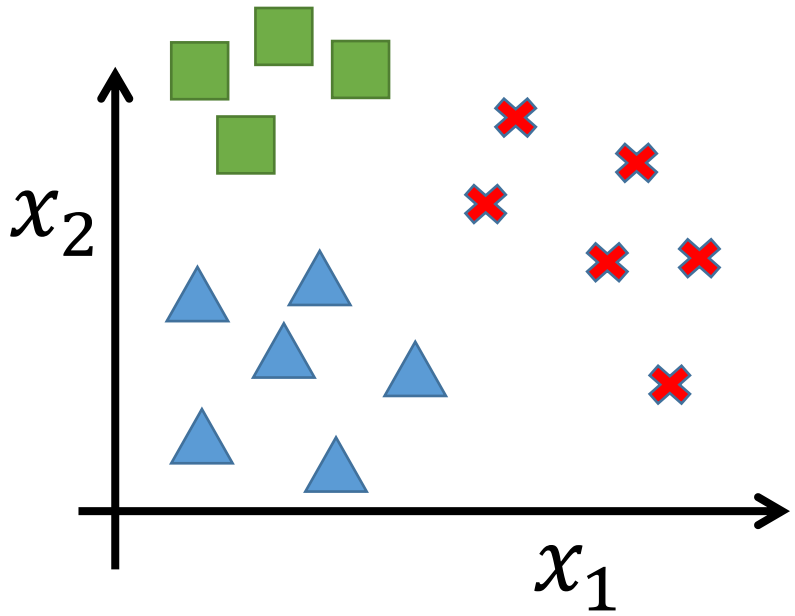


Find $(K-1)K/2$ classifiers $f_{(1,2)}, f_{(1,3)}, \dots, f_{(K-1,K)}$

- $f_{(1,2)}$ classifies $1\ vs\ 2$
- $f_{(1,3)}$ classifies $1\ vs\ 3$
- ...
- $f_{(K-1,K)}$ classifies $K-1\ vs\ K$

- Approach 2: one-versus-rest
- Approach 3: discriminant functions

One-vs-all (one-vs-rest)



Class 1:

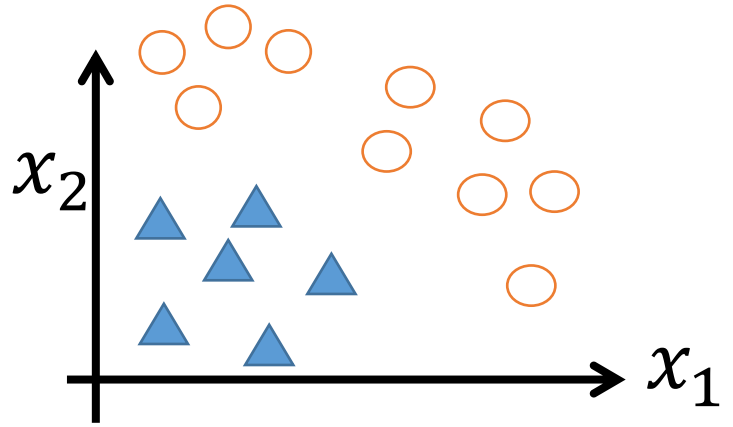
Class 2:

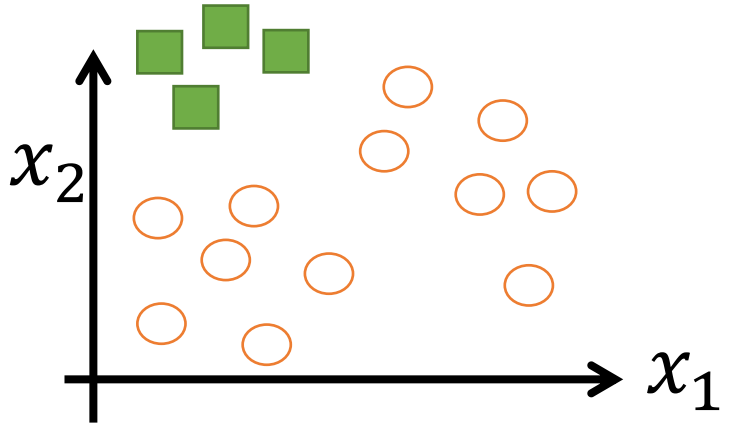
Class 3:

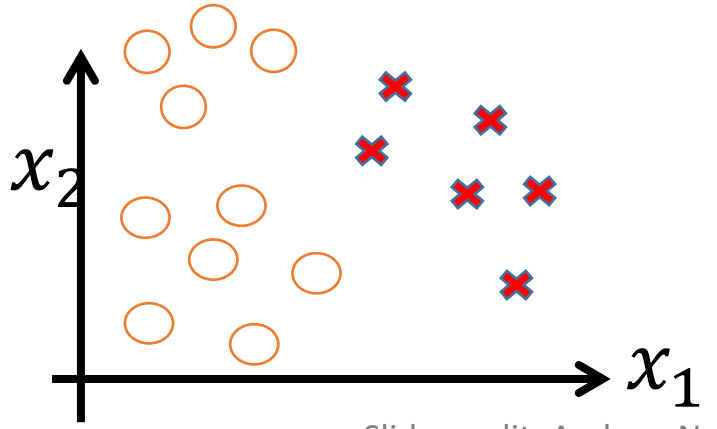
$h_\theta^{(1)}(x)$

$h_\theta^{(2)}(x)$

$h_\theta^{(3)}(x)$







$$h_\theta^{(i)}(x) = P(y = i|x;\theta) \quad (i = 1, 2, 3)$$

Slide credit: Andrew Ng

One-vs-all

- Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$

- Given a new input x, pick the class i that maximizes

$$\max_{\mathrm{i}} h_\theta^{(i)}(x)$$

A way to squash $a = (a_1, a_2, \dots, a_i, \dots)$ into probability vector p

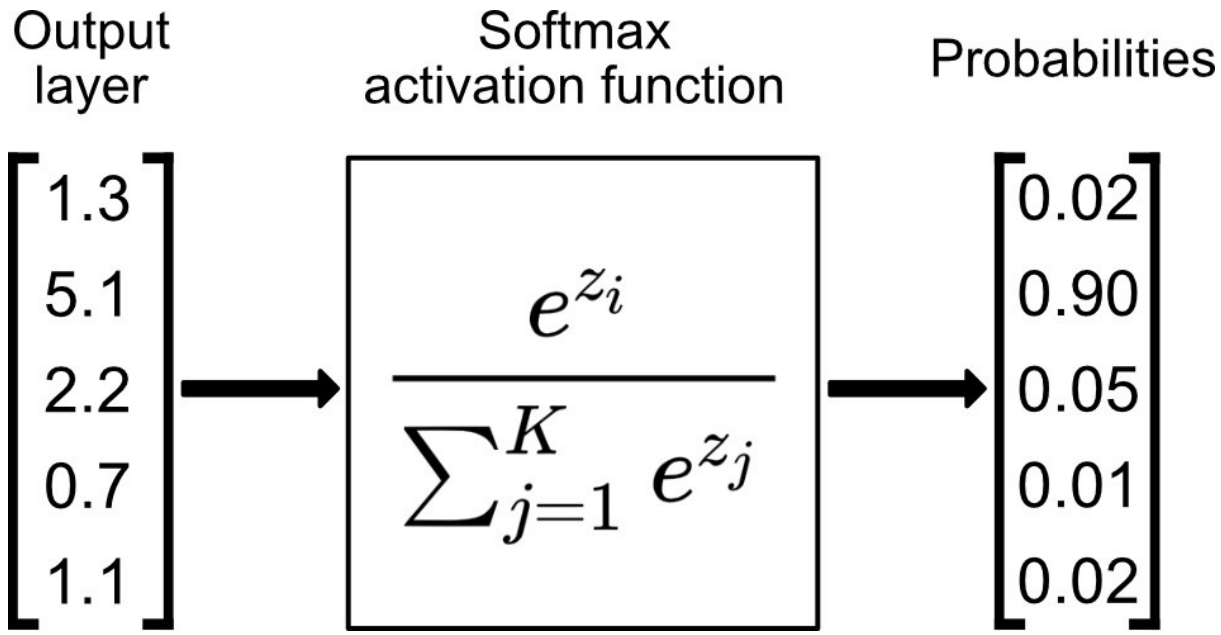
$$\mathrm{softmax}(a) = \left(\frac{\exp(a_1)}{\sum_j \exp(a_j)}, \frac{\exp(a_2)}{\sum_j \exp(a_j)}, \dots, \frac{\exp(a_i)}{\sum_j \exp(a_j)}, \dots \right)$$

SoftMax

A way to squash $a = (a_1, a_2, \dots, a_i, \dots)$ into probability vector p

$$\text{softmax}(a) = \left(\frac{\exp(a_1)}{\sum_j \exp(a_j)}, \frac{\exp(a_2)}{\sum_j \exp(a_j)}, \dots, \frac{\exp(a_i)}{\sum_j \exp(a_j)}, \dots \right)$$

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



SoftMax

Read more on the difference between Softmax and Sigmoid (6 min):

https://dataaspirant.com/difference-between-softmax-function-and-sigmoid-function/