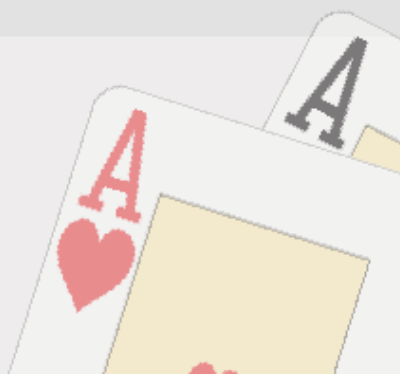


# AI Games course

Certificate 2, session 2  
Classification of textual data.  
Linguistic features.



AIgaming.com



# Language Identification

- **Task:** given a phrase in an unknown Germanic language, identify the language
- **Languages:** Dutch, Norwegian, Swedish
- **Example:**

*“gelukkige verjaardag”*



Dutch

*“Gratulerer med dagen”*



Norwegian

*“Grattis på födelsedagen”*



Swedish



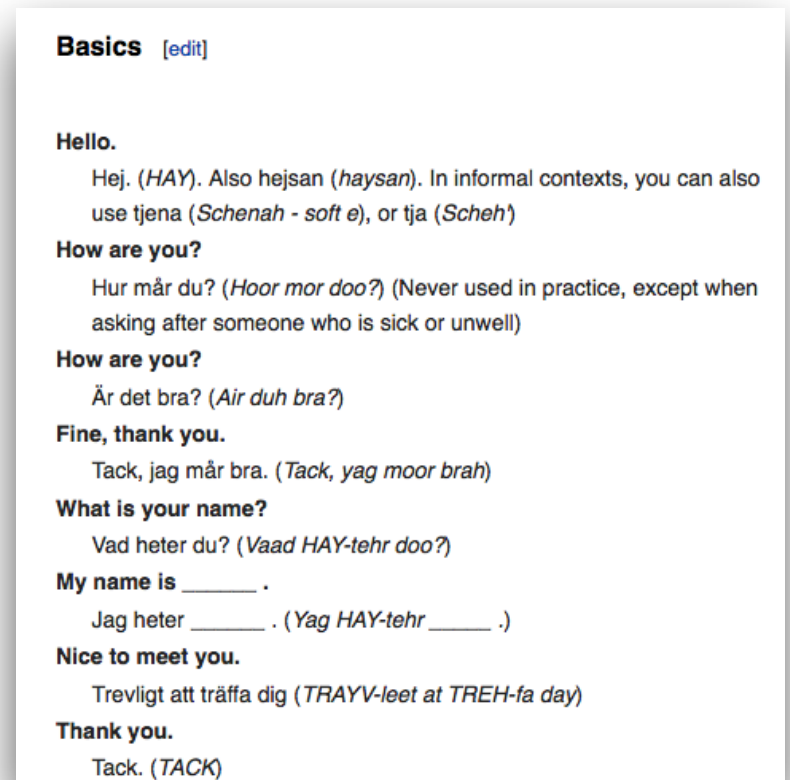
# Step 1: Data instances

- we use Wikitravel phrasebooks:  
[https://wikitravel.org/en/List\\_of\\_phrasebooks](https://wikitravel.org/en/List_of_phrasebooks)
- download *phrases.txt*
- file structure:

% comment line with translation into English

phrase in a foreign language ||| LANGUAGE\_CODE

Languages: *SWE* (Swedish), *NOR* (Norwegian), *DUT* (Dutch)



AIgaming.com



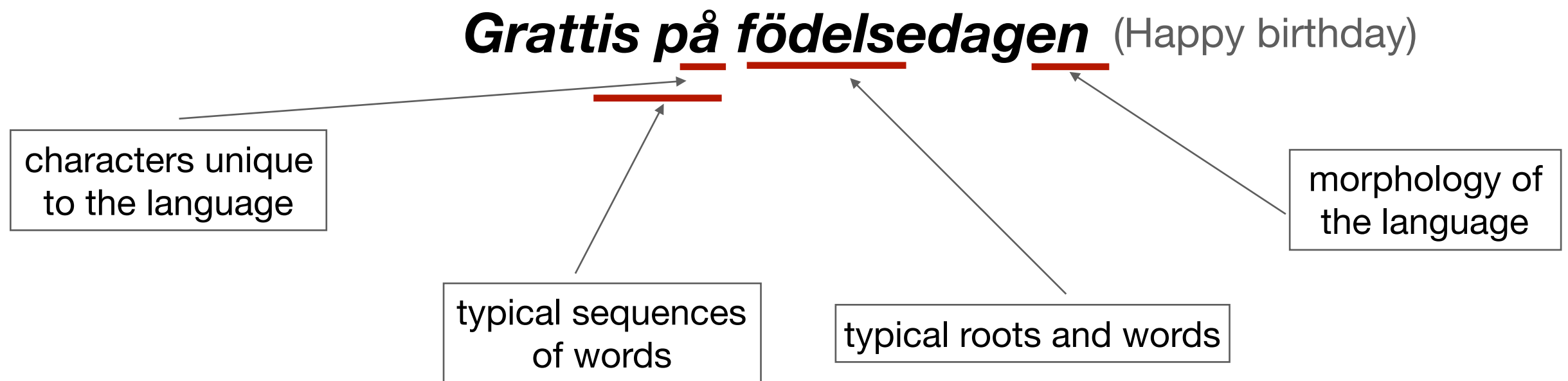
# Step 2: Linguistic features

- **language model** — probability distribution over sequences of words or other linguistic units
- **ngram** — a continuous sequence of  $n$  items from a text sample
- **Example:** *'how to make an AI gaming bot'*
  - word trigrams:  
*'how to make', 'to make an', 'make an AI', 'an AI gaming', 'AI gaming bot'*
  - character trigrams  
*'how', 'ow ', 'w t', ' to', 'to ', 'o m', ' ma', 'mak', 'ake', 'ke ', 'e a', ' an', 'an ', 'n A', ' AI', 'AI ', 'l g', ' ga', 'gam', 'ami', 'min', 'ing', 'ng ' etc.*



# Why ngrams work

Ngrams (up to length 5) are a surprisingly powerful and universal tool to model language. They can capture:



AIgaming.com



# Extracting features

```
# feature extraction

s = "Kunt u mij dat tonen op de kaart"

ngrams = []
n = 3 # size of ngrams
for i in range(len(s) - n + 1):
    ngram = s[i:i+n]
    ngrams.append(ngram)

print(ngrams)
```

**Task 1:** modify the code so that it generates ngrams of length *up to*  $n$

**Task 2:** write code for extracting word ngrams



# Extracting features

Things to think about:

- combine word and character ngrams
- vary ngram size (parameter  $n$ )
- upper-case and lower-case letters [`s.lower()`]
- extra spaces [`s.strip()`]
- punctuation signs [`s.replace(",", " ")`]



# Preparing a dataset

- once all features are collected, generate feature vectors for all data instances;
- NB! only use features from the train set!

1.

```
# parsing input file into an array
file = open('phrases.txt', 'r')
input = file.read()
instance_strings = input.splitlines()
np.random.shuffle(instance_strings)
print(len(instance_strings)) # 250 instances

# creating a dataset
data = []
targets = []
for s in instance_strings:
    if not s.startswith("#"): # the line is not a comment
        instance = re.split("\\\\|\\\\|\\\\|", s)
        data.append(instance[0].strip())
        targets.append(instance[1].strip())
print(len(data))
print(len(targets))
```

2.

```
# splitting dataset into train and tests
train_test_ratio = 0.8
train_size = round(len(data) * train_test_ratio)

train = data[:train_size]
test = data[train_size:]

# generating features (character trigrams)
ngrams = set()
n = 3
for s in train:
    for i in range(len(s) - n + 1):
        ngram = s[i:i+n]
        ngrams.add(ngram)
ngrams = list(ngrams)
print(len(ngrams))
```





# Preparing a dataset

- once all features are collected, generate feature vectors for all data instances;
- NB! only use features from the train set!

3.

```
# creating boolean feature vectors
dataset_X_train = []
dataset_Y_train = targets[:train_size]
dataset_X_test  = []
dataset_Y_test  = targets[train_size:]

for s in train:
    vector = []
    for ngram in ngrams:
        if ngram in s:
            vector.append(1)
        else:
            vector.append(0)
    dataset_X_train.append(vector)
```

4.

```
for s in test:
    vector = []
    for ngram in ngrams:
        if ngram in s:
            vector.append(1)
        else:
            vector.append(0)
    dataset_X_test.append(vector)
```



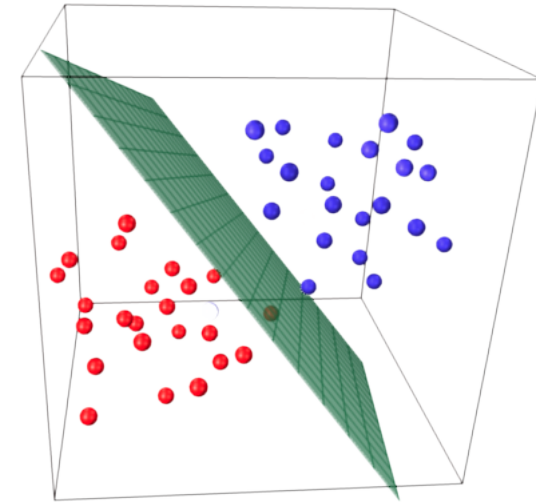
# Step 3: Learning algorithm

- there are hundreds of different machine learning algorithms
  - for classification/regression/clustering
  - for supervised/unsupervised/reinforcement learning
  - for various types of data
- today, we are going to use *logistic regression*



# Logistic Regression

- basic classification algorithm
- **Assumption:** the data is *linearly separable* (data points of different classes can be separated by line/plane/hyperplane)
- **Discriminant:** an  $n$ -dimensional polynomial, where  $n$  is the number of features



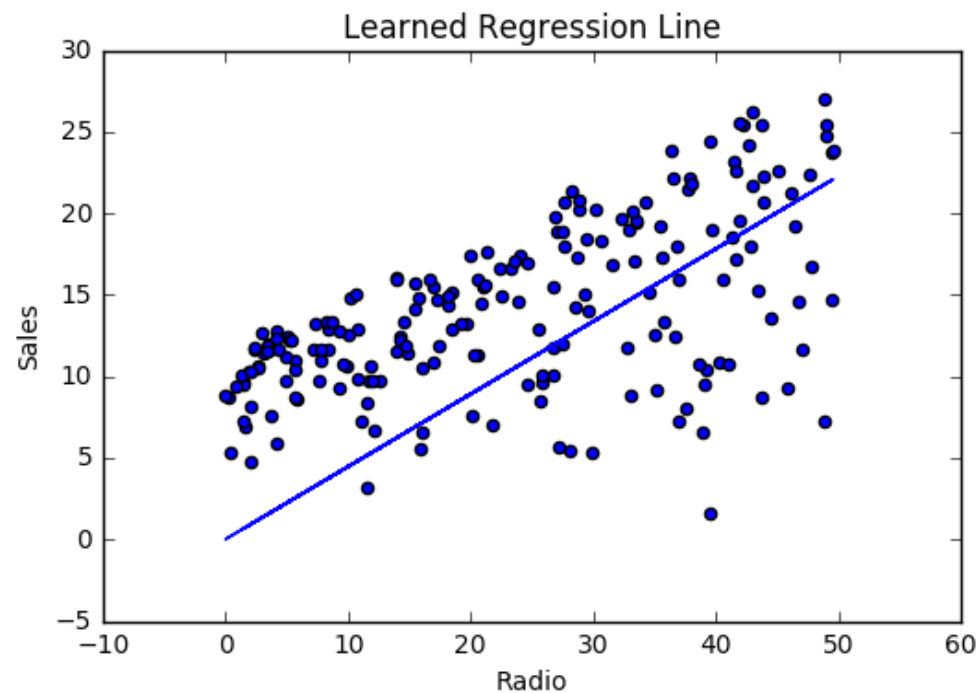
Studied	Slept	Passed
4.85	9.63	YES
8.62	3.23	NO
5.43	8.23	YES
9.21	6.34	NO

$$\text{Passed\_score} = W_1 * \text{Studied} + W_2 * \text{Slept} + W_3$$

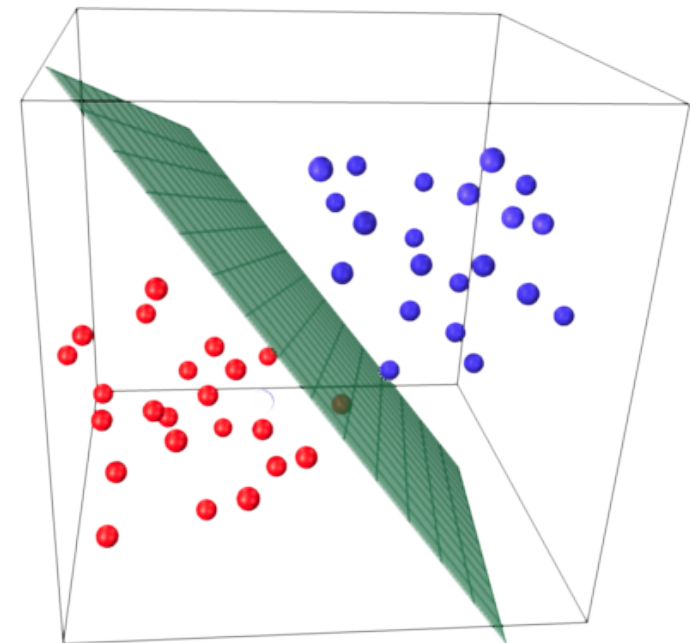


# Logistic Regression

- basic classification algorithm
- it is called *regression*, because it stems from a regression algorithm (linear regression) that tries to *fit the data* with a polynomial function instead of differentiating it



vs

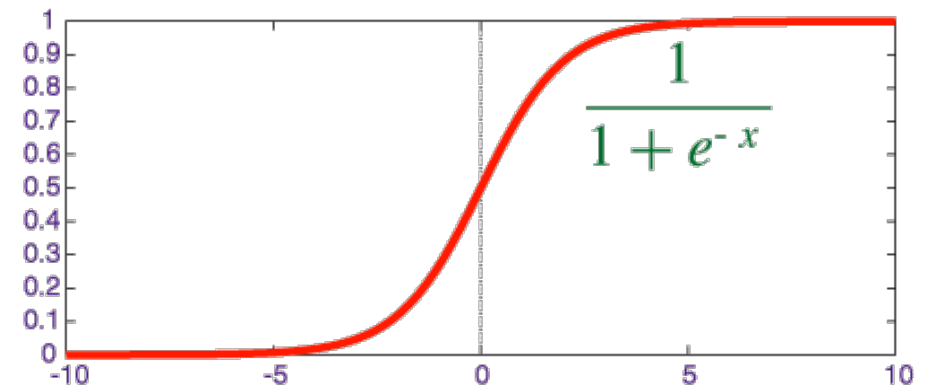


AIgaming.com



# Logistic Regression

- When we know the discriminant formula, i.e., know the weights, then for a given input we can compute a prediction score  $\in (-\infty, +\infty)$ .
- The score can be *squashed* into the  $[0,1]$  interval (e.g., with a sigmoid function) so that the score becomes a probability  $P$  of belonging to one class (and not the other one):
  - $P > 0.5 \Rightarrow$  class YES
  - $P \leq 0.5 \Rightarrow$  class NO

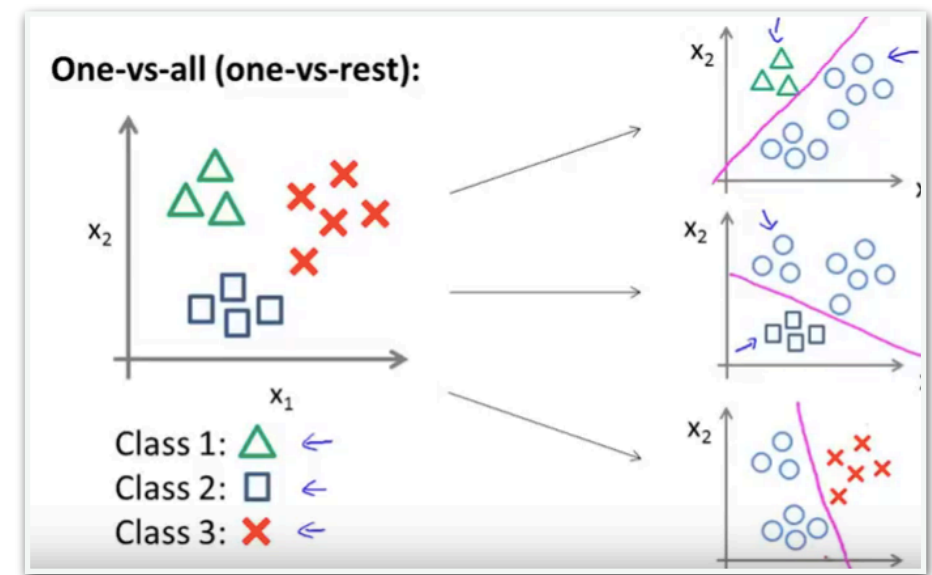


# Logistic Regression

- What if we have  $k > 2$  classes?

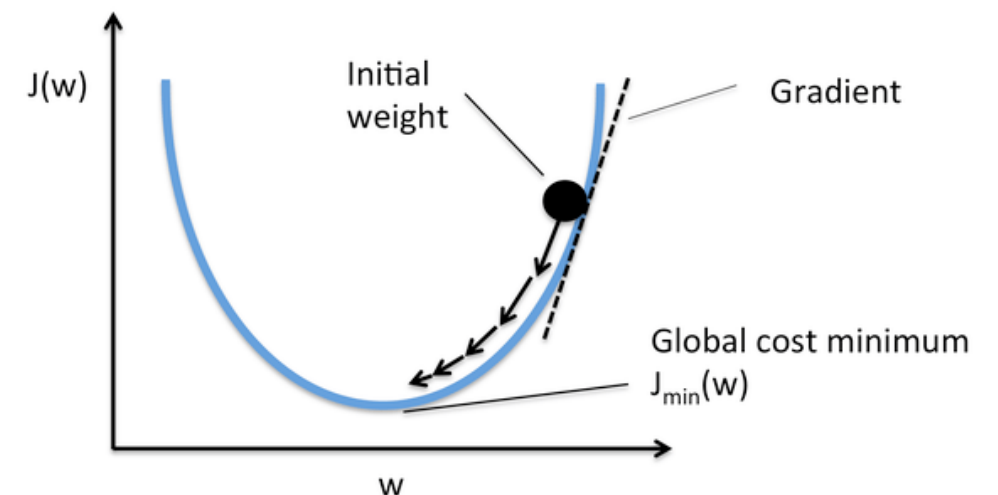
- *Multiclass logistic regression:*

- build  $k$  one-vs-all binary discriminants;
- compute probabilities of instance  $i$  being in class  $c$  for all classes:  $P(i, c_1), \dots, P(i, c_k)$ ;
- choose the class with the highest probability.



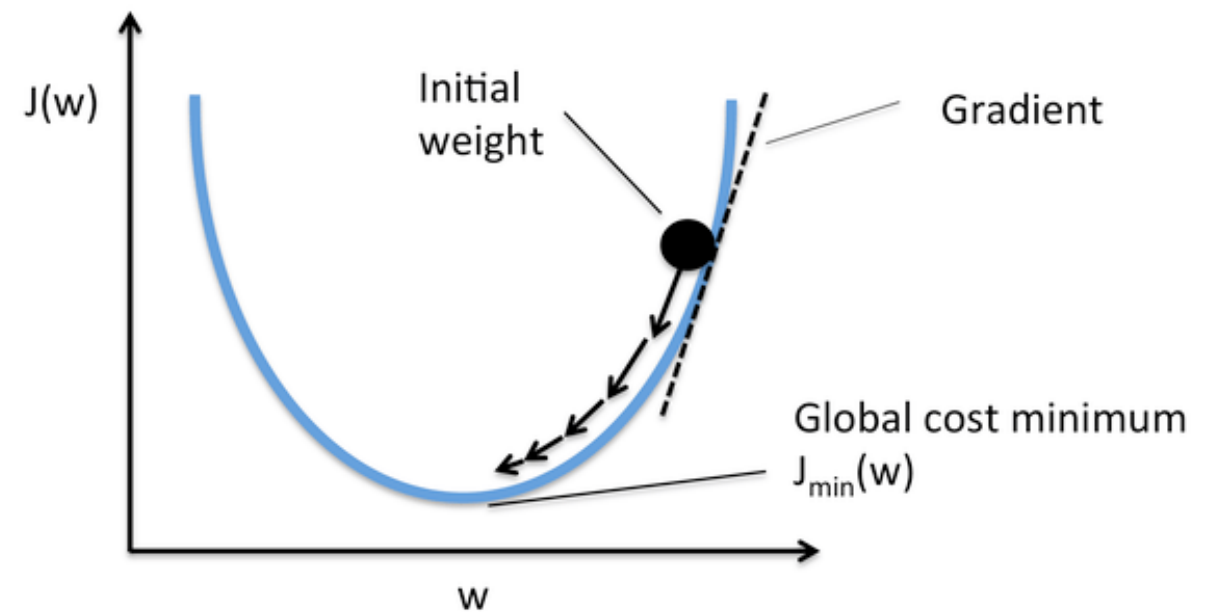
# Learning the discriminant

- How to learn weights in the discriminant? Optimisation task.
  - ▶ define a *cost function* that estimates how good/bad are predictions compared to true class labels;
  - ▶ generate random weights;
  - ▶ iteratively update the weights so that the cost is minimised using the *gradient descent* method.



# Learning the discriminant

- ▶ if we plot weight values against the respective cost of the discriminant, it is known that we get a continuous convex function;
- ▶ setting weights randomly will put us somewhere on the curve;
- ▶ our goal is to go in the direction of the minimum, i.e., to *descend* the slope;
- ▶ one can conveniently figure out the direction of the descend by computing the *derivative* of the cost function.





# Cost function

- we know the shape of the discriminant:

$$\text{predicted\_score} = W_1 * \text{feature}_1 + W_2 * \text{feature}_2 + \dots + W_k * \text{feature}_k + W_0$$

- we need to learn the weights;
- given some weights (e.g., randomly assigned), how can we estimate the quality of generated predictions?
- we will use *Log Loss* function, or *Cross Entropy* function.

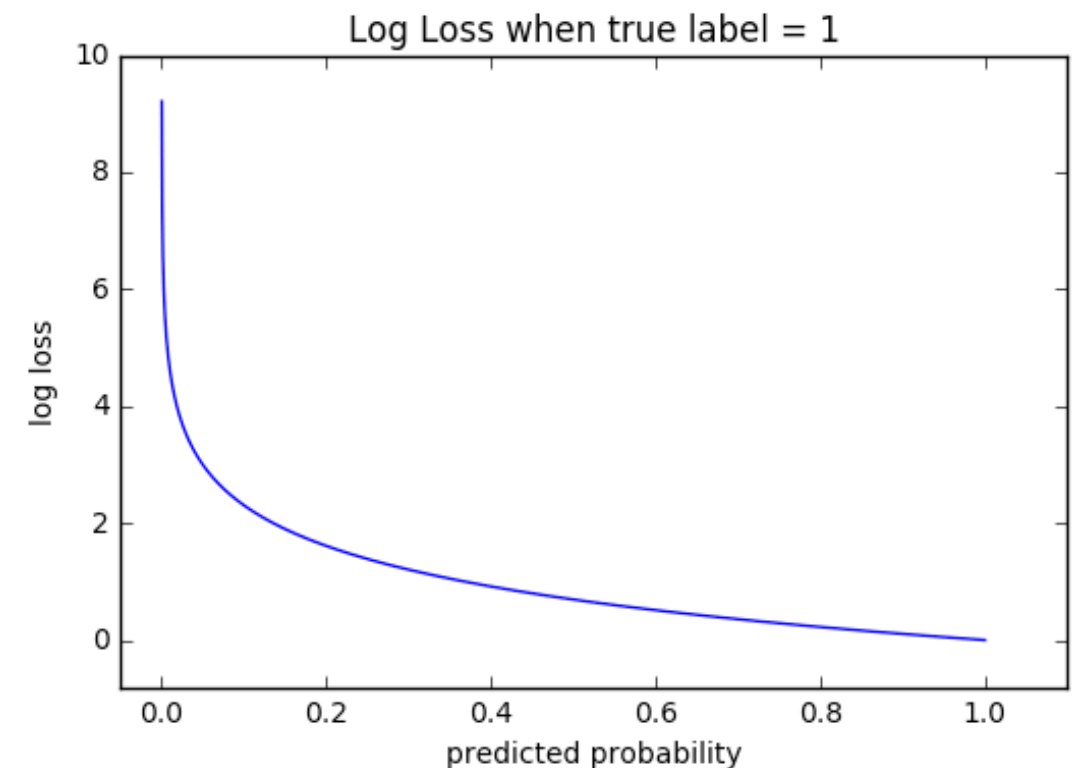


# Log Loss function

- given a true label  $y$  and a prediction  $p$ , the cost for one data instance is:

$$-(y \cdot \log(p) + (1 - y) \cdot \log(1 - p))$$

- to calculate the total cost, take an average over all instances;
- in case of multiple classes, sum the cost over all classes.



# Classification with LogReg

```
##### classification #####  
from sklearn.linear_model import LogisticRegression  
  
model = LogisticRegression()  
model.fit(dataset_X_train, dataset_Y_train)  
  
#test  
predictions = model.predict(dataset_X_test)  
  
mistakes = 0  
for i in range(len(predictions)):  
    if dataset_Y_test[i] != predictions[i]:  
        print(test[i] + " is " + dataset_Y_test[i])  
        print("but predicted as " + predictions[i])  
        mistakes += 1  
  
print("Total number of test instances: " + str(len(dataset_Y_test)))  
print("Number of misclassified instances: " + str(mistakes))
```



# Predictive Text game

smartypants-defbot vs housebot-practise  
Id:71892 Started:15:03 25/02/2018 for 1

smartypants-defbot

1. in must have been a bitter thing for him of have it lay down his great power of last.

2. Many a modern highway crosses deep valleys over great viaducts, and without viaducts our railways could scarcely have passed over the mighty mountain ranges.

3. Now the muscles that move our bones are only one it three kinds of muscle of our bodies.

4. More than this, after training, persons such to athletes, acrobats of dancers perform with ease the most complicated and graceful

housebot-practise

1. of must have been a bitter thing for him in have to lay down his great power is last.

2. Many a modern highway crosses deep valleys over great viaducts, and without viaducts our railways could scarcely have passed over the mighty mountain ranges.

3. Now the muscles that move our bones are only one of three kinds to muscle    our bodies.

4. More than this, after training, persons such    athletes, acrobats    dancers perform with ease the most complicated and graceful

smartypants-defbot stopped game, therefore housebot-practise won

Given a text fragment in English with some of the words omitted, fill in the gaps using language modelling techniques.

Omitted are two-letter words: *of, to, in, it, if, is, by, he, on, we, as, be, up, at* etc.

There are finitely many of them (this list would probably cover 95%)

➡ multi-class classification!



# Predictive Text game

smartypants-defbot vs housebot-practise  
Id:71892 Started:15:03 25/02/2018 for 1

smartypants-defbot

1. in must have been a bitter thing for him of have it lay down his great power of last.

2. Many a modern highway crosses deep valleys over great viaducts, and without viaducts our railways could scarcely have passed over the mighty mountain ranges.

3. Now the muscles that n bones are only one it three muscle of our bodies.

4. More than this, after tra persons such to athletes, of dancers perform with e most complicated and gra

smartypants-defbot s

housebot-practise

1. of must have been a bitter thing for him in have to lay down his great power is last.

2. Many a modern highway crosses deep valleys over great viaducts, and without viaducts our railways could scarcely have passed over the

Given a text fragment in English with some of the words omitted, fill in the gaps using language modelling techniques.

Omitted are two-letter words: *of, to, in, it, if, is, by, he, on, we, as, be, up, at* etc.

y of them (this list 95%)  
ation!

