UE 803 - Data Science for NLP

Lecture 13: Classification

Claire Gardent - CNRS / LORIA

Previous Lectures

Unsupervised learning

- Clustering
  Grouping documents

- Lexical Semantics
  Creating word representations, Grouping words

- Topic Models
  Grouping documents, discovering topics, dimension reduction

Supervised learning

- Regression
  Predict a value

Today's lecture

Supervised learning

- Classification
  Predict a class

Classification

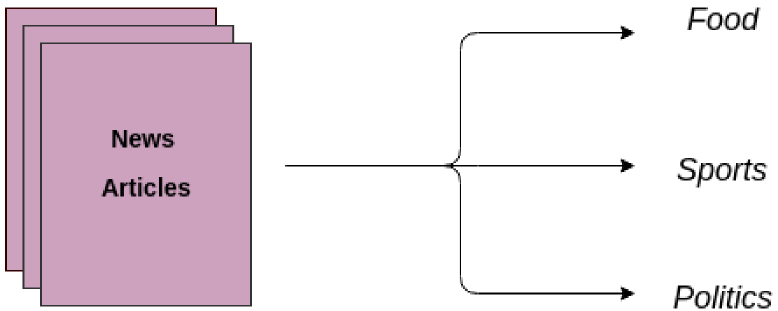
- Introduction
  - Classification: Task and Applications
  - Main Classification Algorithms
  - Machine Learning: Reminder and Terminology
- Features
- Train and Predict
- Evaluation
- Overfitting and generalisation

What is Classification ?

- Classification is a ***Supervised*** Machine Learning algorithm
- Learns to ***classify*** the input i.e. to map the input to a ***class***
- The set of target classes (labels) is pre-specified
  - Example 1: Classifying a mail into spam or not spam (2 classes, ***binary*** classification)
  - Example 2: Classifying a text as talking about Sport, Art or Finance (3 classes, ***N-ary*** classification)

Applications

- Blogs
  - Recommendation
  - Spam filtering
  - Sentiment analysis for marketing
- Newspaper Articles
  - Topic based categorization



- Emails
  - Rerouting
  - Spam filtering
  - Advertising
- General Writing
  - Authorship detection
- Genre detection

Classification Algorithms

Classification Algorithms: Lazy vs Eager learners

Lazy

- No training. At test time, look for most similar instance in stored training data.
- Examples: k-nearest neighbor, Case-based reasoning

Eager

- Learn a model on the training data and use it to predict a class at test time.
- Examples: Decision Tree, Naive Bayes, Artificial Neural Networks

Classification Algorithms: Generative vs. Discriminative

Generative

- Learn a model of the joint probability $p(x, y)$
- Examples: Naives Bayes, Markov Random Fields, Hidden Markov Models, Bayesian Networks

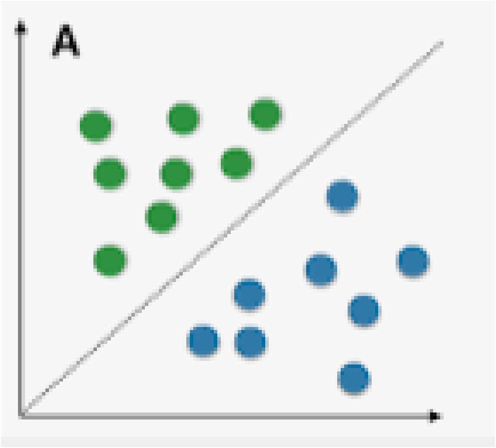
Discriminative

- Learn a conditional probablity $p(y|x)$
- Examples: Logistic Regression, Neural Networks, Nearest Neighbour, Conditional Random Fields

Classification Algorithms: Linear vs. Non Linear

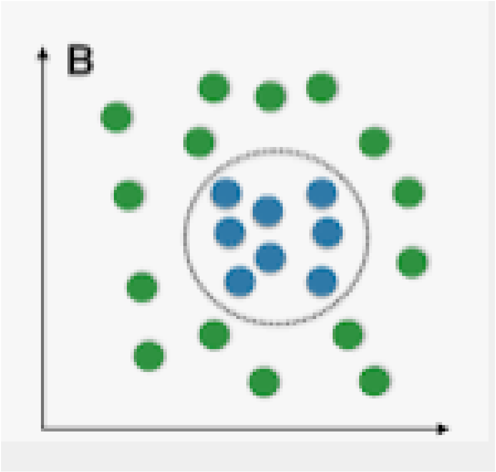
Linear

- When the data is linearly separable
- Perceptron, Logistic regression, ...



Non Linear

- Support Vector Machines
- Neural Networks
- Naives Bayes



Machine Learning Reminder

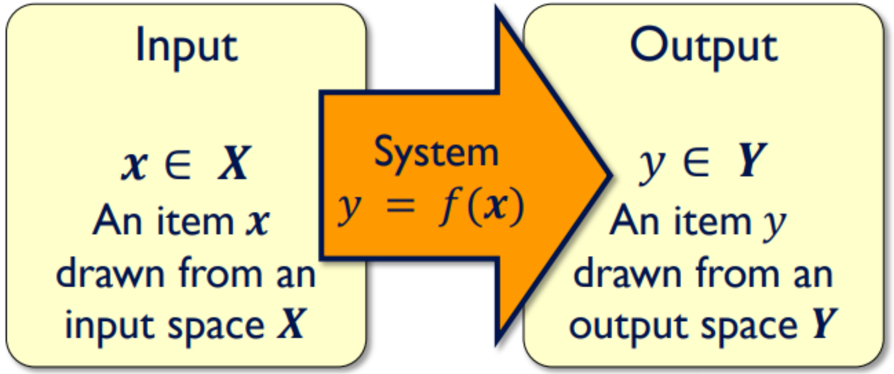
Goal

Given some ***training data***, ***ML algorithms*** aim to learn a function (**model**) *g which maps an* **input** *to an* **output**_

- Regression: the output is a ***numerical (continuous) value***

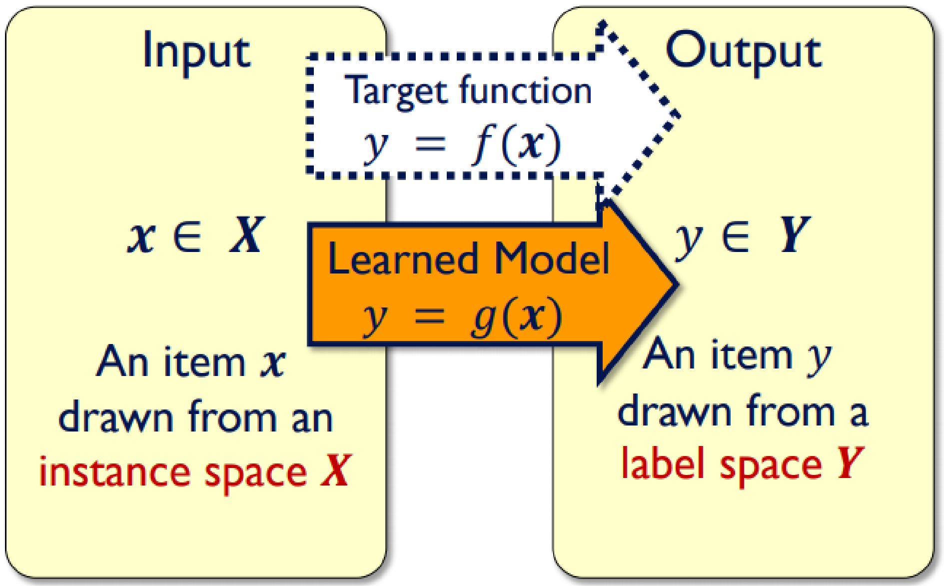
- Classification: the output is a ***class***

- ***Classes*** are also called labels, predictions, outcome, target classes



Training

- The training data consists of input/output pairs related by the target function f
- The function/model g is learned by iterating over (input,output) examples, comparing predicted ($g(x)$) and expected ($f(x)$) results, computing a ***loss*** and using this loss to ***optimise*** (the parameters of) the model g.
- g can then be used to make predictions about new examples (***unseen/text data***)



Features

- The input is represented by one or more ***features*** (multivariate data)
- Features are individual, measurable attributes which caracterise the input data
- They are ***manually defined*** and ***automatically derived*** from the input
- They should ***help the ML algorithm relate input (e.g., text) to output (e.g., text topic)***
- Therefore they depend on your application (sentiment analysis, spam detection etc.)

Example features

- words
- punctuation signs
- n-grams
- part-of-speech tags
- semantic roles
- parse tree based features
- electronic dictionary based features (WordNet)
- cluster identifier (from clustering)
- topic model topic label ...

Features

- ML algorithms manipulate numbers, not text
- Features must be converted to numbers
  - Categorical features (male/female, black/white/red etc) are converted using so-called ***1-hot vectors***.
    E.g., black = <0,0,1>, white = <0,1,0> , red = <11,0,0>
  - Continuous features are binned (group of ages, of long/medium/short sentences)

Each input is represented by a vector of features/numbers

Categorical features

- Words/tokens are categorical features. They are converted to numbers using a vector whose size is the size of the vocabulary.
- The ***vocabulary*** is the set of (distinct) tokens occurring in the data.
- token2idx and idx2token dictionaries are used to be able to go back and forth between natural language token and their ML number representation

Data

- John eats an apple. The apple is ripe.

Vocabulary

- {John, eats, an, apple, The, is, ripe}

One-hot vector for "The"

- <0,0,0,0,1,0,0>

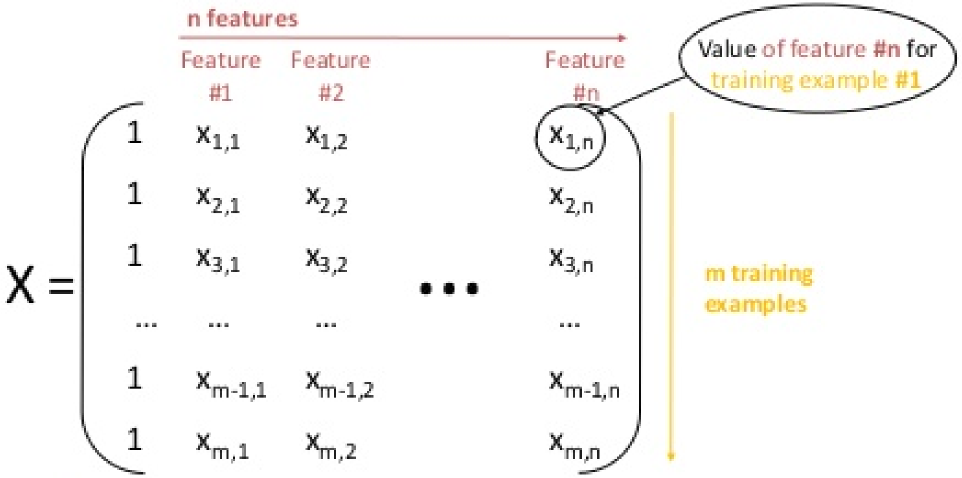
One-hot vector for " The apple is ripe"

- <0,0,0,1,1,1,1>

token2idx["The"] = 4
idx2token[4] = "The"

Training Data

- Each input is converted to a ***feature vector***
- The size of the feature vector is the number of features used
- The components of the feature vector indicate the presence/absence (for a binary feature) or the frequency of the feature in the data point



With m examples/data points/observation and n features, the input data is represented by a matrix X ***with shape*** (m, n)

The matrix Y contains the class assigned to each input. It is of ***shape*** $(m, 1)$]

Learning a Model

ML algorithms learn a function $g(x) = \hat{y}$ that can map input variables to output variables

- x: a vector of features
  These are ***given***

- w: a vector of ***parameters*** (weights) of the model
  These are ***learned***

- Machine Learning algorithms automatically ***learn the importance (weight) of each feature*** .

- The ***score function*** is the inner product between the weights and the features (= the weighted sum of the input features)

The Perceptron Classifier

Classifying Mails into Spam or Not Spam

Input

- Mail

Perceptron Output

- $y \in -1, 1$
- -1: negative class
  e.g., not spam
- 1: Positive class
  e.g., spam

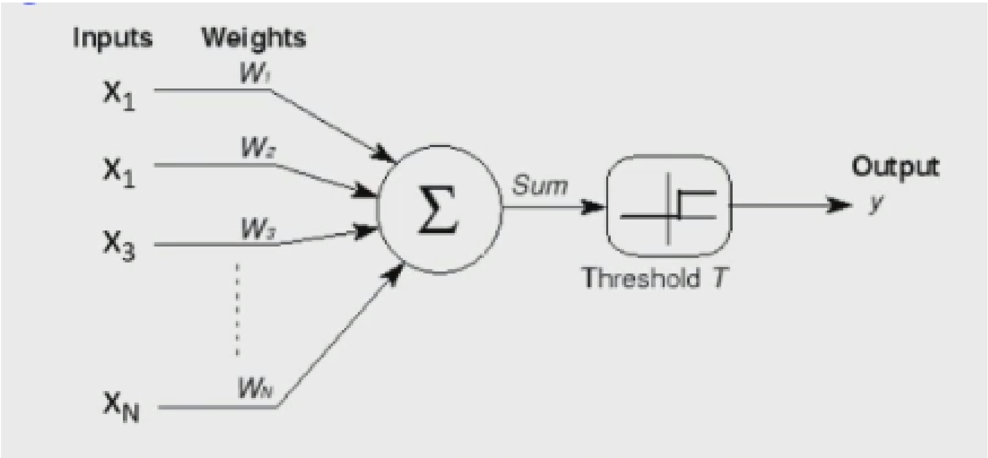
Computing the Perceptron Output

Given the current parameter values w_i and the input x with features $f_i(x)$,

the Perceptron computes the weighted sum of the features

$$A_w(x) = \sum_i w_i \times f_i(x)$$

If $A_w(x)) \geq 0$:
then the output is 1 (spam)
else the output is -1 (not spam)



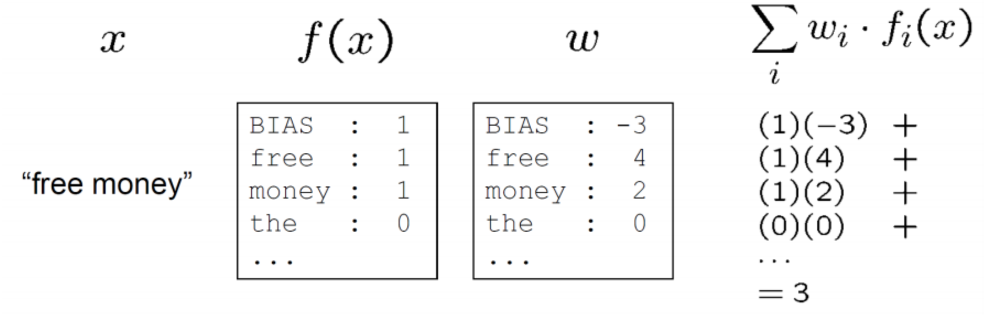
Computing the Perceptron Output

Given the current parameter values w_i and the input x with features $f_i(x)$,

the Perceptron computes the weighted sum of the features

$$A_w(x) = \sum_i w_i \times f_i(x)$$

If $A_w(x)) \geq 0$:
then the output is 1 (spam)
else the output is -1 (not spam)



$A_w(x) = 3 \geq 0$

So the output is 1 and the mail ("free money") is classified as spam.

How are the parameters learned ?

- Start with random weights

- Iterate over the training instances:

  - Classify the instance with the current weights
  - If the prediction is correct ($y = \hat{y}$):
    - go to the next training instance
  - Else:
    - ***modify the weights*** (= learning) using the perceptron update rule

Perceptron Update Rule

$$w \leftarrow w + \eta(y_i * x_i)$$

w is the weight vector

x_{ij} is feature vector for sample (input) i

y_i is the target (correct) output for sample i

$\hat{y_i}$ is the Perceptron output for sample i

η is the learning rate (a small constant between 0 and 1)

Weight are only updated when $\hat{y_i} \neq y_i$

Intuition behind the update

Suppose we have made a mistake on a positive example. Then,
$y_i = 1, \hat{y_i} = -1, w * x_i < 0$

Update rule
$w \leftarrow w + \eta(y_i * x_i)$
$\Leftrightarrow w \leftarrow w + x_i$
(assuming $\eta = 1$ and since $y_i = 1$)

New score $\hat{y_i} = (w + x_i) * x_i$

The updated weights help increase the score bringing it closer to 1, the expected value

Multiclass Perceptron

E.g., Classify news report into sport, politics or technology

- Estimate a weight vector w_c for each class
- Compute the activation for each class

$$A_w(x, c) = \sum_i w_{c,i} \times f_i(x)$$

- The class with highest activation is the predicted class

$$c = argmax_c(A_w(x, c))$$

Multiclass Perceptron

"win the vote"

```
BIAS  :  1
win   :  1
game  :  0
vote  :  1
the   :  1
...
```

w_{SPORTS}

```
BIAS  : -2
win   :  4
game  :  4
vote  :  0
the   :  0
...
```

$w_{POLITICS}$

```
BIAS  :  1
win   :  2
game  :  0
vote  :  4
the   :  0
...
```

w_{TECH}

```
BIAS  :  2
win   :  0
game  :  2
vote  :  0
the   :  0
...
```

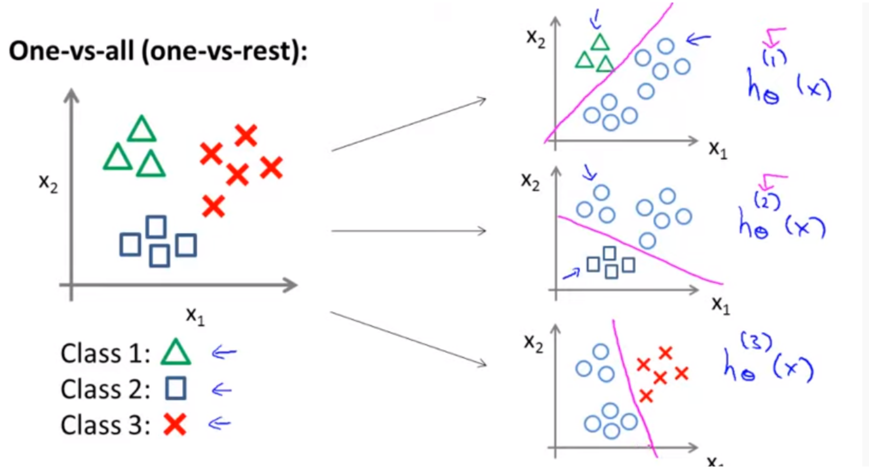
Logistic Regression

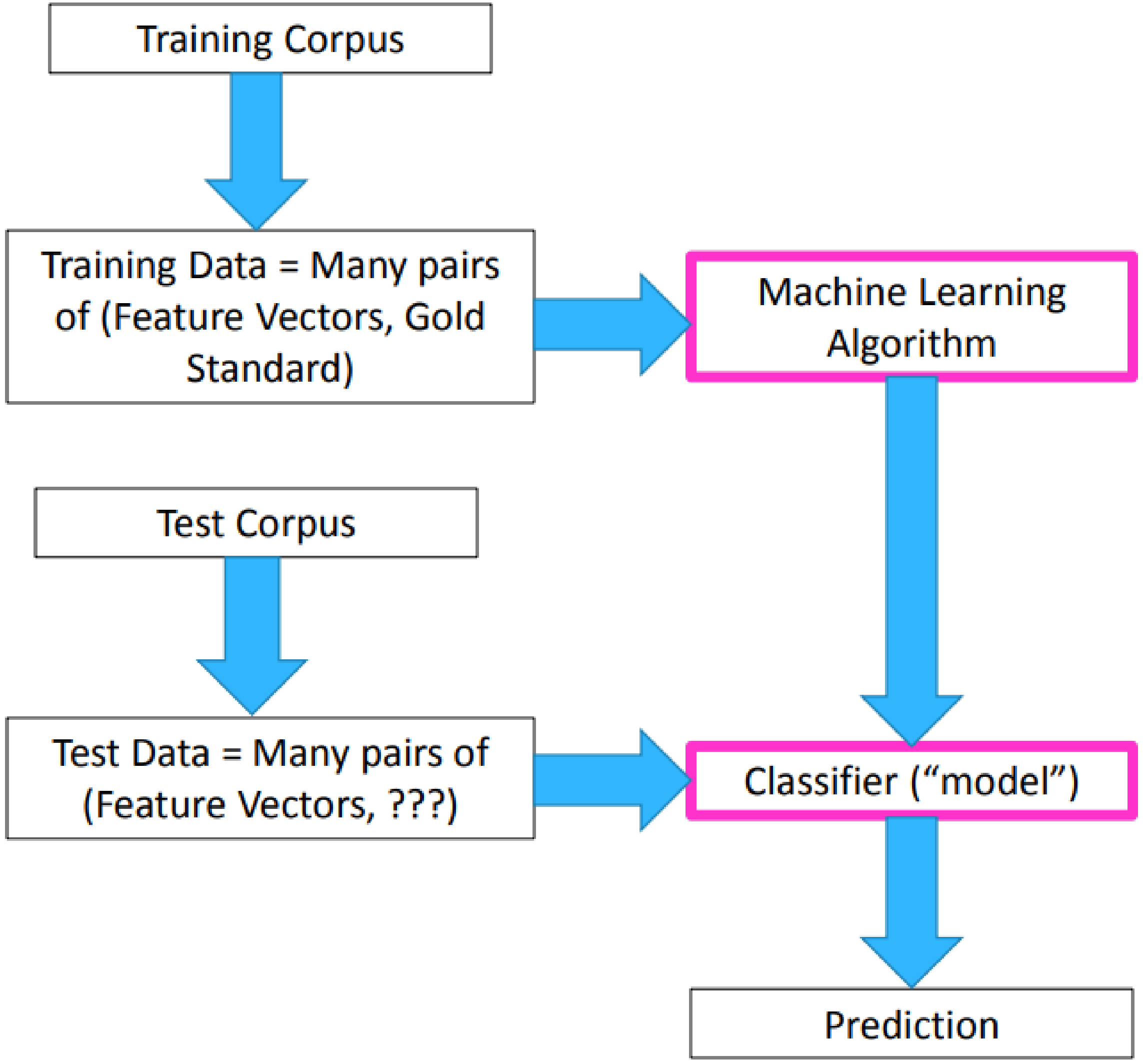
- Predicts the ***probability*** of an instance belonging to a class
- Uses the ***logistic function*** (σ) to determine this probability
- IF $A_w(x) \geq 50\%$, output = 1, else output = 0
  (1 = positive class, 0 = negative class)

$$A_w(x) = \sigma(\Sigma_i w_i \times x_i) = \frac{1}{1 + e^{-\Sigma_i w_i \times x_i}}$$

]

- Multi class
  - Train a classifier one-vs-all for each class
  - Predict class with highest probability (whose classifier is most confident)

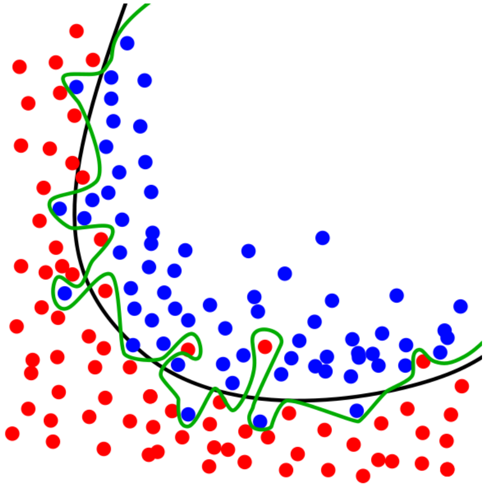



Training Corpus
Training Data = Many pairs of (Feature Vectors, Gold Standard)
Machine Learning Algorithm
Test Corpus
Test Data = Many pairs of (Feature Vectors, ???)
Classifier (“model”)
Prediction

Overfitting

"The **green line represents an overfitted model** and the **black line represents a regularized model**. While the green line best follows the data, it is too dependent on the training data" (Mohri)

- The ML algorithm fits its model too closely to the training data. It ***memorizes*** the data and does not learn to ***generalise***. Will not not perform well on previously unseen data



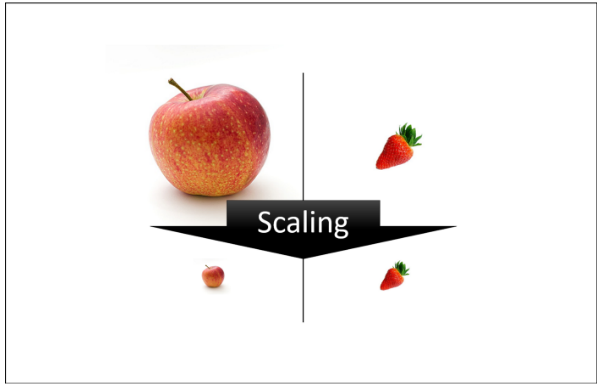
- Overfitting may result from using too many features, not having enough training data and/or running too many iterations on the training data

- Regularisation can be used to help reduce overfitting

Feature Scaling

- Different features can have very different ranges
- Large differences in values between different features are not always meaningful
- Scaling is used to ensure that feature values belong to the same range.

Name	Weight	Price
Orange	15	1
Apple	18	3
Banana	12	2
Grape	10	5



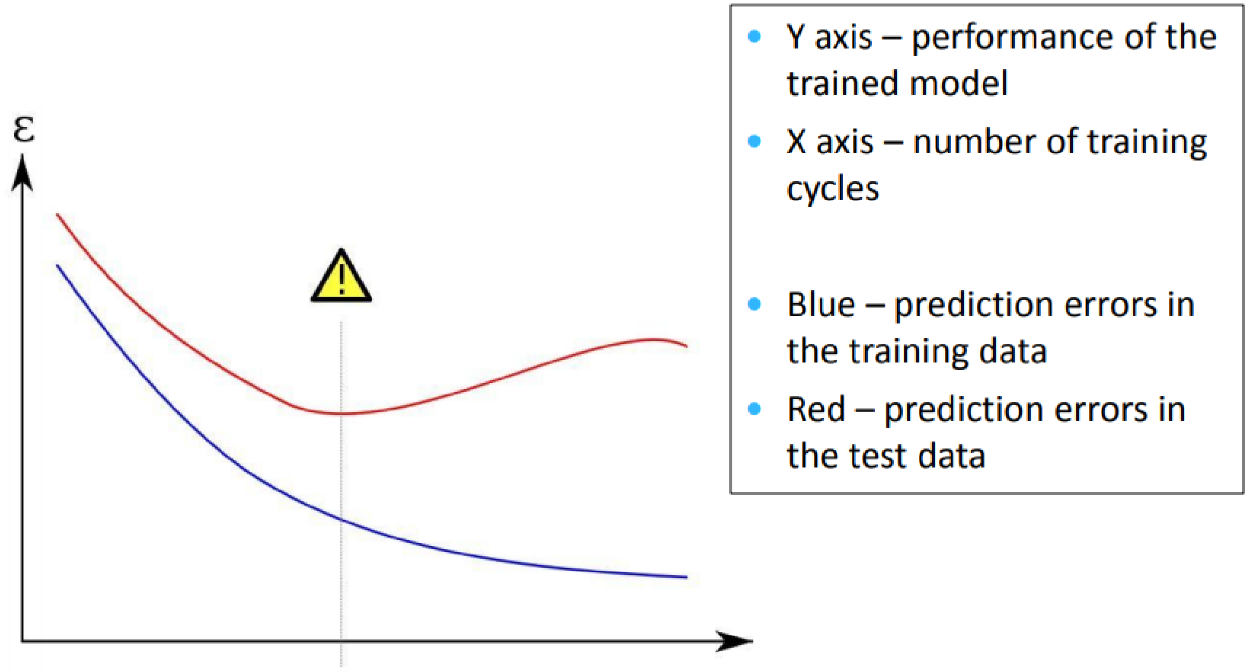
Feature Scaling

- There are multiple ways of scaling features (min max, standard, max abs etc).
- A standard way to scale features is Z-score normalisation (also called standardization) which ensures that features are normally distributed ($\mu = 0, \rho = 1$).
- μ and ρ are computed on the training data (not the test data)
- Each feature value x is scaled as:

$$Z = \frac{x - \mu}{\rho}$$

- Standardisation is applied to both the training data and the test data

Overfitting



The performance is high on the training data but low on test (previously unseen) data

Accuracy, Precision and Recall

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Accuracy

$$A = \frac{(TP + TN)}{(TP + FP + FN + TN)}$$

- Fraction of instances predicted correctly
- Only use when classes are balanced

Precision

$$P = \frac{TP}{TP + FP}$$

Recall

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

F1-Score

$$F1 = 2 * \frac{P \times R}{P + R}$$

Example

	Classified positive	Classified negative
Positive class	0 (TP)	25 (FN)
Negative class	0 (FP)	125 (TN)

A classifier which always predicts the negative class (e.g., not spam)

Accuracy = 125/150 = 0.83

$$A = \frac{(TP + TN)}{(TP + FP + FN + TN)}$$

Precision = 0

$$P = \frac{TP}{TP + FP}$$

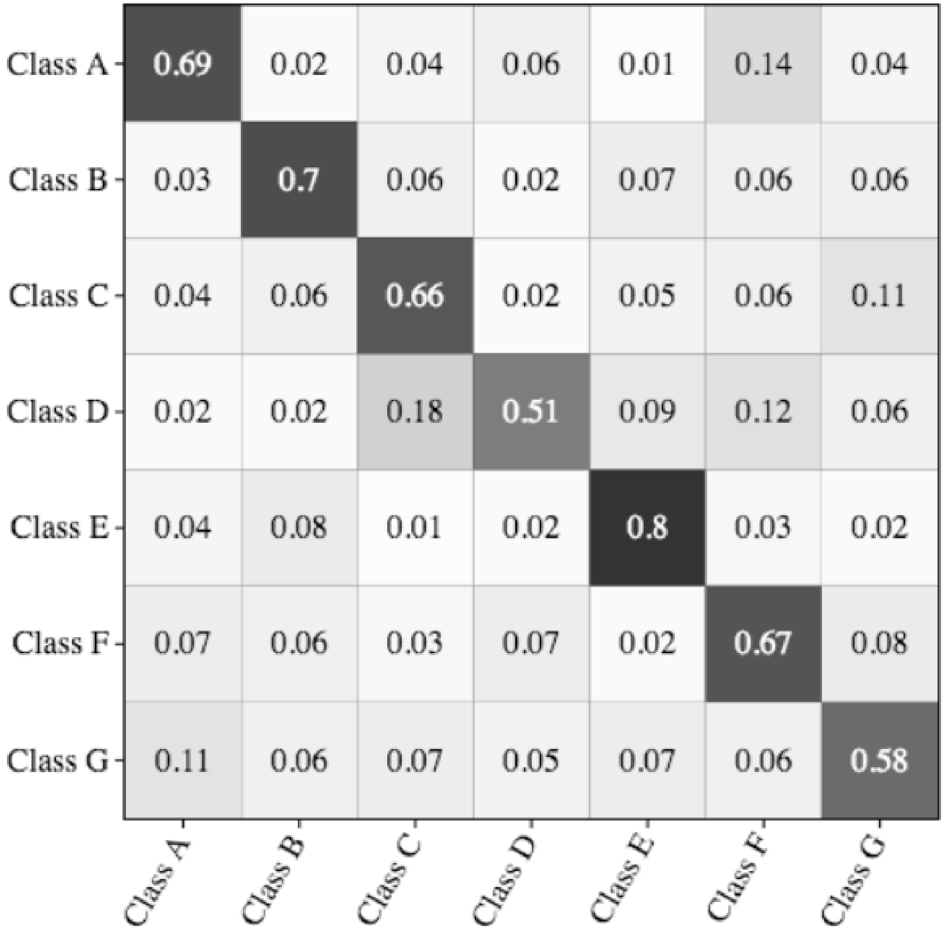
Out of all the examples the classifier labeled as positive, what fraction were correct?

Recall = 0

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

Out of all the positive examples there were, what fraction did the classifier pick up?

Confusion Matrix



Classification in Python

Splitting the data into train and test

```
# Import 'train_test_split'
from sklearn.model_selection import train_test_split

# Shuffle and split the data into training and testing subsets
# Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,
                                                    test_size=0.20, random_state=42)
```

Converting the input data to features using Scikit-learn tf-idf vectorizer

```
# Using TFIDF vectorizer to convert convert words to Vector Space
tfidf_vectorizer = TfidfVectorizer(max_features=8000,
                                   use_idf=True,
                                   stop_words='english',
                                   tokenizer=nltk.word_tokenize,
                                   ngram_range=(1, 3))

# Fit the vectorizer to train and test data
X_train_vec = tfidf_vectorizer.fit_transform(X_train)
X_test_vec = tfidf_vectorizer.transform(X_test)

features = tfidf_vectorizer.get_feature_names()
print(features)
```

`fit_transform` computes the scaling parameters (μ, ρ) on the training data and scales the training data accordingly.

`transform` scales the test data using the scaling parameters computed on the training data.

Train, test and Evaluate

```
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

# Create a Perceptron object
classifier = Perceptron(max_iter=5)

# Train the model on the training data
classifier.fit( X_train_vec, Y_train )

# Test the model on the test data
Y_pred = classifier.predict( X_test_vec )

# Print out the expected values and the predictions
print( '\nExpected Values:',  Y_test )
print( '\nPredictions:',  Y_pred )

# Print accuracy
print( "Acc:", accuracy_score( Y_test, Y_pred) )
# Print the confusion matrix
print( confusion_matrix(Y_test, Y_pred ) )
```

Useful Links

- Scikit-learn documentation on data transformations and specifically the API for the classes CountVectorizer and TfidfTransformer
- Source: Lecture Slides on Classification
- Short Video on Logistic Regression
- Blog on various types of classifiers
- StatQuest Video on Logistic Regression
- Fasttext Blog with Code
- Code with Explanation
- Fast.ai Video on Classification