# Statistical Natural Language Processing

Classification

Çağrı Çöltekin

University of Tübingen
Seminar für Sprachwissenschaft

Summer Semester 2020



## When/why do we do classification

- Is a given email spam or not?
- What is the gender of the author of a document?
- Is a product review positive or negative?
- Who is the author of a document?
- What is the subject of an article?
- …

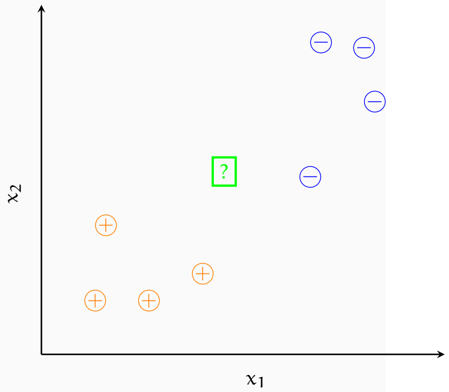
As opposed to regression, the outcome is a 'category'.





## The task

- Given a set of training data with (categorical) labels
- Train a model to predict future data points from the same distribution







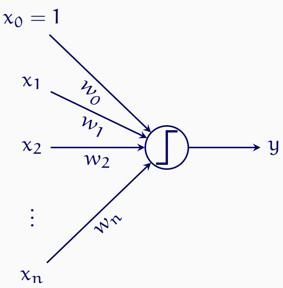

## Outline

- Perceptron
- Logistic regression
- Naive Bayes
- Multi-class strategies for binary classifiers
- Evaluation metrics for classification
- Brief notes on what we skipped





## The perceptron




$$y = f\left(\sum_i^n w_i x_i\right)$$

where

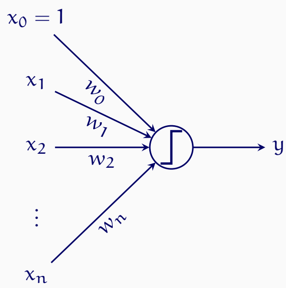
$$f(x) = \begin{cases} +1 & \text{if } \sum_i^n w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

Similar to the *intercept* in linear models, an additional input $x_0$ which is always set to one is often used (called *bias* in ANN literature)





## The perceptron: in plain words




- Sum all input $x_i$ weighted with corresponding weight $w_i$
- Classify the input using a threshold function

positive the sum is larger than 0
negative otherwise





## Learning with perceptron

- We do not update the parameters if classification is correct
- For misclassified examples, we try to minimize

$$E(w) = -\sum_i \boldsymbol{w}\boldsymbol{x}_i y_i$$

where $i$ ranges over all misclassified examples
- Perceptron algorithm updates the weights such that

$$w \leftarrow w - \eta \nabla E(w)$$
$$w \leftarrow w + \eta \boldsymbol{x}_i y_i$$

for misclassified examples. $\eta$ is the learning rate



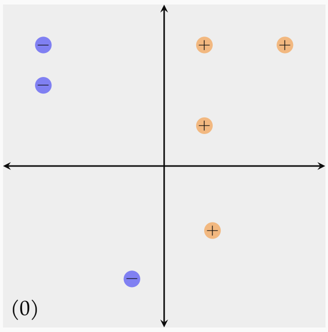


## The perceptron algorithm

- The perceptron algorithm can be

online update weights for a single misclassified example
batch updates weights for all misclassified examples at once

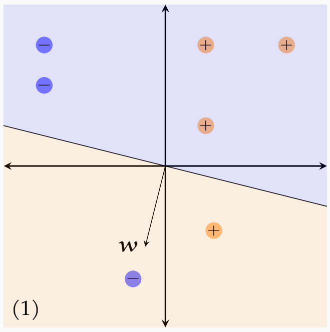
- The perceptron algorithm converges to the global minimum if the classes are *linearly separable*
- If the classes are not linearly separable, the perceptron algorithm will not stop
- We do not know whether the classes are linearly separable or not before the algorithm converges
- In practice, one can set a stopping condition, such as
  - Maximum number iterations/updates
  - Number of misclassified examples
  - Number of iterations without improvement




1. Randomly initialize $\boldsymbol{w}$ the decision boundary is orthogonal to $\boldsymbol{w}$
2. Pick a misclassified example $\boldsymbol{x}_i$ add $y_i\boldsymbol{x}_i$ to $\boldsymbol{w}$
3. Set $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i\boldsymbol{x}_i$, go to step 2 until convergence



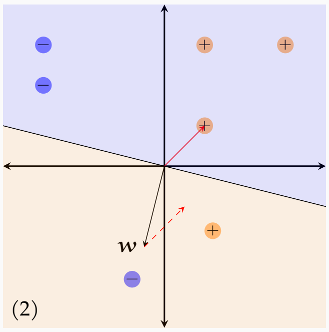





1. Randomly initialize $\boldsymbol{w}$ the decision boundary is orthogonal to $\boldsymbol{w}$
2. Pick a misclassified example $\boldsymbol{x}_i$ add $y_i\boldsymbol{x}_i$ to $\boldsymbol{w}$
3. Set $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i\boldsymbol{x}_i$, go to step 2 until convergence



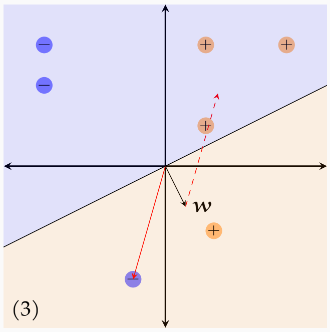





1. Randomly initialize $\boldsymbol{w}$ the decision boundary is orthogonal to $\boldsymbol{w}$
2. Pick a misclassified example $\boldsymbol{x}_i$ add $y_i\boldsymbol{x}_i$ to $\boldsymbol{w}$
3. Set $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i\boldsymbol{x}_i$, go to step 2 until convergence



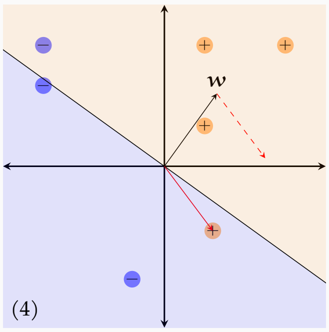





1. Randomly initialize $\boldsymbol{w}$ the decision boundary is orthogonal to $\boldsymbol{w}$
2. Pick a misclassified example $\boldsymbol{x}_i$ add $y_i\boldsymbol{x}_i$ to $\boldsymbol{w}$
3. Set $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i\boldsymbol{x}_i$, go to step 2 until convergence



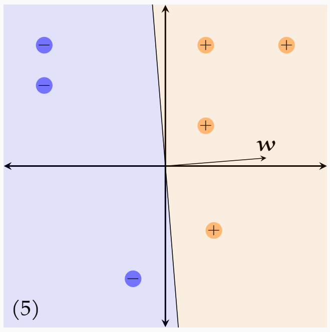





1. Randomly initialize $\boldsymbol{w}$ the decision boundary is orthogonal to $\boldsymbol{w}$
2. Pick a misclassified example $\boldsymbol{x}_i$ add $y_i\boldsymbol{x}_i$ to $\boldsymbol{w}$
3. Set $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i\boldsymbol{x}_i$, go to step 2 until convergence








1. Randomly initialize $\boldsymbol{w}$ the decision boundary is orthogonal to $\boldsymbol{w}$
2. Pick a misclassified example $\boldsymbol{x}_i$ add $y_i\boldsymbol{x}_i$ to $\boldsymbol{w}$
3. Set $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i\boldsymbol{x}_i$, go to step 2 until convergence





## Perceptron: a bit of history

- The perceptron was developed in late 1950's and early 1960's (Rosenblatt 1958)
- It caused excitement in many fields including computer science, artificial intelligence, cognitive science
- The excitement (and funding) died away in early 1970's (after the criticism by Minsky and Papert 1969)
- The main issue was the fact that the perceptron algorithm cannot handle problems that are not linearly separable



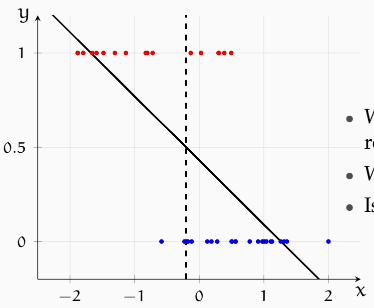


## Logistic regression

- Logistic *regression* is a *classification* method
- In logistic regression, we fit a model that predicts $P(y \mid x)$
- Logistic regression is an extension of linear regression
  - it is a member of the family of models called generalized linear models
- Typically formulated for binary classification, but it has a natural extension to multiple classes
- The multi-class logistic regression is often called *maximum-entropy model* (or max-ent) in the NLP literature

## Data for logistic regression

an example with a single predictor




- Why not just use linear regression?
- What is $P(y \mid x = 2)$?
- Is RMS error appropriate?





## Fixing the outcome: transforming the output variable

- The prediction we are interested in is $\hat{y} = P(y = 1|\mathbf{x})$
- We transform it with logit function:

$$\text{logit}(\hat{y}) = \log \frac{\hat{y}}{1 - \hat{y}} = w_0 + w_1 x$$

- $\frac{\hat{y}}{1-\hat{y}}$ (odds) is bounded between 0 and $\infty$
- $\log \frac{\hat{y}}{1-\hat{y}}$ (log odds) is bounded between $-\infty$ and $\infty$
- we can estimate $\text{logit}(\hat{y})$ with regression, transform with the inverse of logit()

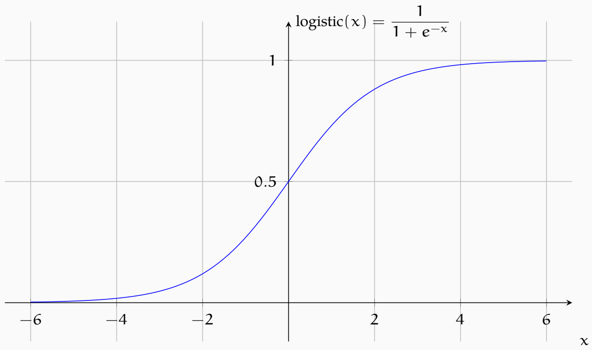
$$\hat{y} = \frac{e^{w_0 + w_1 x}}{1 + e^{w_0 + w_1 x}} = \frac{1}{1 + e^{-w_0 - w_1 x}}$$

which is called logistic (sigmoid) function





## Logistic function








## How to fit a logistic regression model

with maximum-likelihood estimation

$$P(y = 1|\mathbf{x}) = p = \frac{1}{1 + e^{-\mathbf{w}\mathbf{x}}} \qquad P(y = 0|\mathbf{x}) = 1 - p = \frac{e^{-\mathbf{w}\mathbf{x}}}{1 + e^{-\mathbf{w}\mathbf{x}}}$$

The likelihood of the training set is,

$$\mathcal{L}(\mathbf{w}) = \prod_i p^{y_i}(1 - p)^{1 - y_i}$$

In practice, we maximize log likelihood, or minimize '$-\log$ likelihood':

$$-\log \mathcal{L}(\mathbf{w}) = -\sum_i y_i \log p + (1 - y_i) \log(1 - p)$$





## How to fit a logistic regression model (2)

- Bad news: there is no analytic solution
- Good news: the (negative) log likelihood is a convex function
- We can use iterative methods such as *gradient descent* to find parameters that maximize the (log) likelihood
- Using gradient descent, we repeat

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w})$$

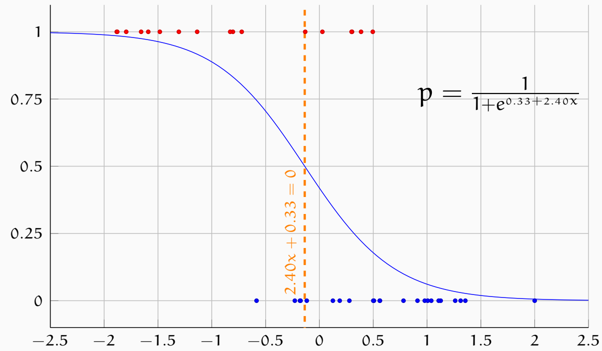
until convergence, $\eta$ is the *learning rate*





## Example logistic-regression

back to the example with a single predictor

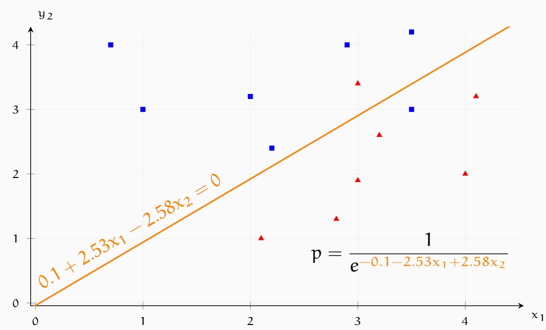







## Another example

two predictors








## Multi-class logistic regression

- Generalizing logistic regression to more than two classes is straightforward
- We estimate,

$$P(C_k \mid x) = \frac{e^{\mathbf{w}_k \mathbf{x}}}{\sum_j e^{\mathbf{w}_j \mathbf{x}}}$$

where $C_k$ is the $k^{th}$ class, j iterates over all classes.

- The function is called the *softmax* function, used frequently in neural network models as well
- This model is also known as *log-linear model*, *maximum entropy* model, or *Boltzmann machine*

## Naive Bayes classifier

- Naive Bayes classifier is a well-known simple classifier
- It was found to be effective on a number tasks, primarily in *document classification*
- Popularized by practical spam detection applications
- *Naive* part comes from a strong independence assumption
- *Bayes* part comes from use of Bayes' formula for inverting conditional probabilities
- However, learning is (typically) 'not really' Bayesian





## Naive Bayes: estimation

- Given a set of features $\mathbf{x}$, we want to know the class $y$ of the object we want to classify
- At prediction time we pick the class, $\hat{y}$

$$\hat{y} = \arg\max_{y} P(y \mid \mathbf{x})$$

- Instead of directly estimating the conditional probability, we invert it using the Bayes' formula

$$\hat{y} = \arg\max_{y} \frac{P(\mathbf{x} \mid y)P(y)}{P(\mathbf{x})} = \arg\max_{y} P(\mathbf{x} \mid y)P(y)$$

- Now the task becomes estimating $P(\mathbf{x} \mid y)$ and $P(y)$





## Naive Bayes: estimation (cont.)

- Class distribution, $P(y)$, is estimated using the MLE on the training set
- With many features, $\mathbf{x} = (x_1, x_2, \ldots x_n)$, $P(\mathbf{x} \mid y)$ is difficult to estimate
- Naive Bayes estimator makes a conditional independence assumption: given the class, we assume that the features are independent of each other

$$P(\mathbf{x} \mid y) = P(x_1, x_2, \ldots x_n \mid y) = \prod_{i=1}^{n} P(x_i \mid y)$$





## Naive Bayes: estimation (cont.)

- The probability distributions $P(x_i \mid y)$ and $P(y)$ are typically estimated using MLE (count and divide)
- A *smoothing* technique may be used for unknown features (e.g., words)
- Note that $P(x_i \mid y)$ can be

binomial e.g, whether a word occurs in the document or not
categorical e.g, estimated using relative frequency of words
continuous the data is distributed according to a known distribution





## Naive Bayes

a simple example: spam detection

Training data:

| features present | label |
|---|---|
| `good book` | NS |
| `now book free` | S |
| `medication lose weight` | S |
| `technology advanced book` | NS |
| `now advanced technology` | S |

- A test instance: {`book`, `technology`}
- Another one: {`good`, `medication`}

$P(S) = 3/5$, $P(NS) = 2/5$

| $w$ | $P(w \mid S)$ | $P(w \mid NS)$ |
|---|---|---|
| `medication` | 1/5 | 0 |
| `free` | 1/5 | 0 |
| `technology` | 1/5 | 1/5 |
| `advanced` | 1/5 | 1/5 |
| `book` | 1/5 | 2/5 |
| `now` | 1/5 | 0 |
| `lose` | 1/5 | 0 |
| `weight` | 1/5 | 0 |
| `good` | 0 | 1/5 |





## Classifying classification methods

another short digression

- Some classification algorithms are non-probabilistic, discriminative: they return a label for a given input. Examples: perceptron, SVMs, decision trees
- Some classification algorithms are discriminative, probabilistic: they estimate the conditional probability distribution $p(\mathbf{c} \mid \mathbf{x})$ directly. Examples: logistic regression, (most) neural networks
- Some classification algorithms are generative: they estimate the joint distribution $p(\mathbf{c}, \mathbf{x})$. Examples: naive Bayes, Hidden Markov Models, (some) neural models





## More than two classes

- Some algorithms can naturally be extended to handle multiple class labels
- Any binary classifier can be turned into a k-way classifier by

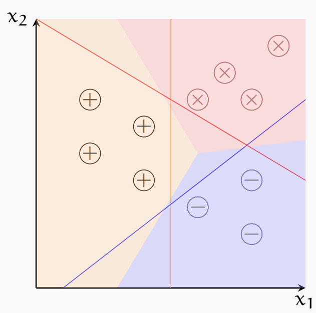
OvR one-vs-rest or one-vs-all
- train k classifiers: each learns to discriminate one of the classes from the others
- at prediction time the classifier with the highest confidence wins
- needs confidence score from the base classifiers

OvO one-vs-one
- train $\frac{k(k-1)}{2}$ classifiers: each learns to discriminate a pair of classes
- decision is made by (weighted) majority vote
- works without need for confidence scores, but needs more classifiers





## One vs. Rest




- For 3 classes, we fit 3 classifiers separating one class from the rest
- Some regions of the feature space will be ambiguous
- We can assign labels based on probability or weight value, if classifier returns one
- One-vs.-one and majority voting is another option

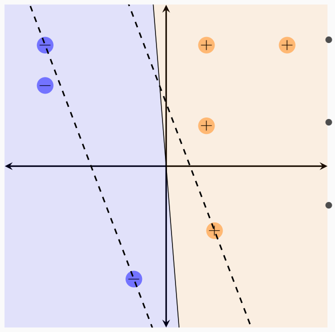
## More classification methods ...

- Classification is a well-studied topic in ML, with a large range of applications
- There are many different approaches
- In most cases you can 'plug' a classification algorithm instead of another, treating classifiers as 'black boxes'
- You should, however, understand the methods you use: you may not be able to use them properly if you do not understand them
- One-slide introduction to some of the methods we did not cover starts on the next slide
- We will return to some specialized methods later in this course





## Maximum-margin methods (e.g., SVMs)

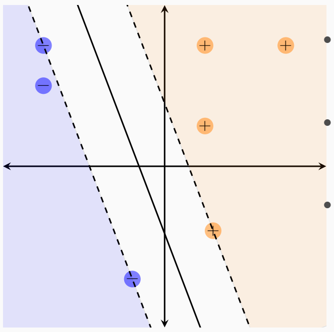


- In perceptron, we stopped whenever we found a linear discriminator
- Maximum-margin classifiers seek a discriminator that maximizes the margin
- SVMs have other interesting properties, and they have been one of the best 'out-of-the-box' classifiers for many problems





## Maximum-margin methods (e.g., SVMs)



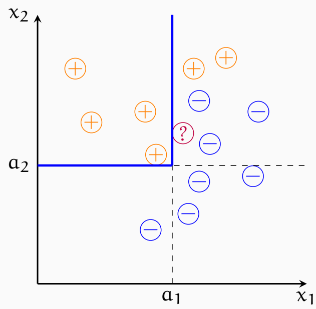
- In perceptron, we stopped whenever we found a linear discriminator
- Maximum-margin classifiers seek a discriminator that maximizes the margin
- SVMs have other interesting properties, and they have been one of the best 'out-of-the-box' classifiers for many problems

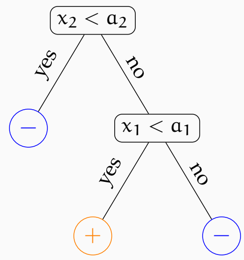




## A quick survey of some solutions

Decision trees



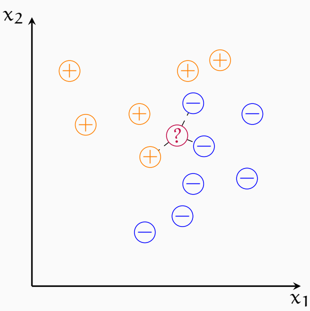

- Note that the decision boundary is non-linear





## A quick survey of some solutions

Instance/memory based methods



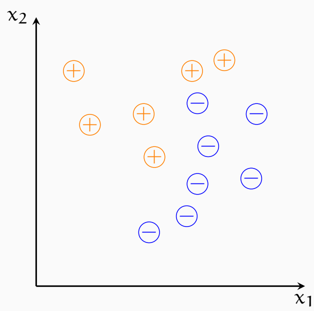

- No training: just memorize the instances
- During test time, decide based on the k nearest neighbors
- Like decision trees, kNN is non-linear
- It can also be used for regression

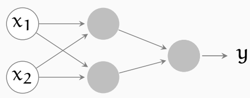




## A quick survey of some solutions

Artificial neural networks








## Measuring success in classification

Accuracy

- In classification, we do not care (much) about the average of the error function
- We are interested in how many of our predictions are correct
- Accuracy measures this directly

$$\text{accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}}$$





## Accuracy may go wrong

- Think about a 'dummy' search engine that always returns an empty document set (no results found)
- If we have
  - 1 000 000 documents
  - 1000 relevant documents (related to the terms in the query)

  the accuracy is:

$$\frac{999\,000}{1\,000\,000} = 99.90\,\%$$

- In general, if our class distribution is *skewed*, of *imbalanced*, accuracy will be a bad indicator of success

## Measuring success in classification

Precision, recall, F-score

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

$$F_1\text{-score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

| | | true value | |
|---|---|---|---|
| | | positive | negative |
| predicted | pos. | TP | FP |
| | neg. | FN | TN |





## Example: back to the 'dummy' search engine

- For a query
  - 1 000 000 documents
  - 1000 relevant documents

$$\text{accuracy} = \frac{999\,000}{1\,000\,000} = 99.90\,\%$$

$$\text{precision} = \frac{0}{1\,000\,000} = 0\,\%$$

$$\text{recall} = \frac{0}{1\,000\,000} = 0\,\%$$

Precision and recall are asymmetric,
the choice of the 'positive' class is important.





## Classifier evaluation: another example

Consider the following two classifiers:

| | | true value | | true value | |
|---|---|---|---|---|---|
| | | positive | negative | positive | negative |
| predicted | pos. | 7 | 9 | 1 | 3 |
| | neg. | 3 | 1 | 9 | 7 |

Accuracy both $8/20 = 0.4$

Precision $7/16 = 0.44$ and $1/4 = 0.25$

Recall $7/10 = 0.7$ and $1/10 = 0.1$

F-score 0.54 and 0.14





## Multi-class evaluation

- For multi-class problems, it is common to report average precision/recall/f-score
- For C classes, averaging can be done two ways:

$$\text{precision}_M = \frac{\sum_i^C \frac{TP_i}{TP_i+FP_i}}{C} \qquad \text{recall}_M = \frac{\sum_i^C \frac{TP_i}{TP_i+FN_i}}{C}$$

$$\text{precision}_\mu = \frac{\sum_i^C TP_i}{\sum_i^C TP_i + FP_i} \qquad \text{recall}_\mu = \frac{\sum_i^C TP_i}{\sum_i^C TP_i + FN_i}$$

($M$ = macro, $\mu$ = micro)

- The averaging can also be useful for binary classification, if there is no natural positive class





## Confusion matrix

- A confusion matrix is often useful for multi-class classification tasks

| | | true class | | |
|---|---|---|---|---|
| | | negative | neutral | positive |
| predicted | negative | 10 | 3 | 4 |
| | neutral | 2 | 12 | 8 |
| | positive | 0 | 7 | 7 |

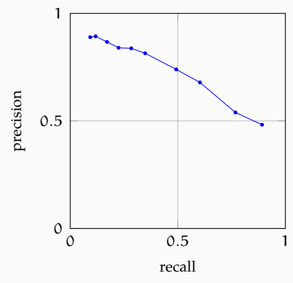
- Are the classes balanced?
- What is the accuracy?
- What is per-class, and averaged precision/recall?





## Precision–recall trade-off

- Increasing precision (e.g., by changing a hyperparameter) results in decreasing recall
- Precision–recall graphs are useful for picking the correct models
- *Area under the curve* (AUC) is another indication of success of a classifier








## Performance metrics a summary

- Accuracy does not reflect the classifier performance when class distribution is skewed
- Precision and recall are binary and asymmetric
- For multi-class problems, calculating accuracy is straightforward, but others measures need averaging
- These are just the most common measures, there are more
- You should understand what these metrics measure, and use/report the metric that is useful for the purpose





## Summary

- We discussed three basic classification techniques: perceptron, logistic regression, naive Bayes
- We left out many others: SVMs, decision trees, …
- We also did not discuss a few other interesting cases, including *multi-label* classification
- We will discuss some (non-linear) classification methods next

Next

Wed ML evaluation, quick summary so far

Mon Introduction to neural networks

# Additional reading, references, credits

- Hastie, Tibshirani, and Friedman (2009) covers logistic regression in section 4.4 and perceptron in section 4.5
- Jurafsky and Martin (2009) explains it in section 6.6, and it is moved to its own chapter (7) in the draft third edition


Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Second. Springer series in statistics. Springer-Verlag New York. ISBN: 9780387848587. URL: `http://web.stanford.edu/~hastie/ElemStatLearn/`.

Jurafsky, Daniel and James H. Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. second. Pearson Prentice Hall. ISBN: 978-0-13-504196-3.

Minsky, Marvin and Seymour Papert (1969). *Perceptrons: An introduction to computational geometry*. MIT Press.

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain.". In: *Psychological review* 65.6, pp. 386–408.