ML2 – the basics

Machine Learning – Tools and applications for policy – Lecture 3

Iman van Lelyveld – Michiel Nijhuis

DNB Data Science Hub

DeNederlandscheBank EUROSYSTEEM DataScience Hub

ML2 – the basics

1. How do I assess success?
    - Confusion matrix, Receiver Operator Characteristic (ROC)
2. What are overfitting, bias and variance?
    - L2-regularisation
3. Support Vector Machines (SVM) and *k* nearest neighbours (KNN)

Outline

- ML2 – the basics
  - Measuring success
  - Overfitting, bias and variance
  - Fixing overfitting
  - Classification: Support Vector Machines (SVM)
  - K-nearest neighbors (KNN)

Knowledge clips

- Evaluation: Simple Measures for Classification (CompStat Munich) (link)

  Focus on 3.12-6.10 for the explanation on using a cost function. The remainder of the clip talks about other cost functions (Brier etc.) – advanced topic.
- Evaluation: Measures for Binary Classification: ROC Measures (CompStat Munich) (link)

  Focus on imbalanced classes at the start (e.g. defaults) and confusing naming (11.39-13.00)
- Evaluation: Measures for Binary Classification: ROC visualization (CompStat Munich) (link)

  If you are interested in deriving ROC/AUC from first principles
- How do Support Vector Machines work? (Brandon Rohrer) (link)
- Decision Boundary (Andrew Ng) (link)
- To discuss in class: Decision tree animation
- To discuss in class: Bias Variance

Outline

Performance evaluation metrics

- So far we've mainly been focussed on accuracy
- Accuracy can be misleading for imbalanced datasets
  - If 99.9% of all days it does not rain, then it will be difficult to beat the very simple predictor: it will be dry tomorrow
- We need to compute the performance for the off-diagonal
  - Is it bad if I predict sunshine but it rains? Is it bad if I predict rain but it is sunny?
  - Is it bad if you test positive for Covid but you are healthy? Is it bad if I say you are healthy but you have Covid?

- Confusion matrix helps visualize different types of errors a classifier can make by reporting the counts of these errors
  - true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions

How to measure performance? – confusion matrix

Accuracy

$$\frac{TP + TN}{TP + FP + FN + TN}$$

Error

$$\frac{FP + FN}{TP + FP + FN + TN}$$

Precision

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

Recall

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

F1 score

$$\mathrm{F1} = 2 \times \frac{Prec \times Rec}{Prec + Rec}$$

Prediction

Actual value	**pos**	**neg**	**Tot.**
pos′	True positive (TP)	False negative (FN)	TP + FN
neg′	False positive (FP)	True negative (TN)	FP + TN
Tot.	TP + FP	FN + TN	

Performance – accuracy

Accuracy

$$\frac{TP + TN}{TP + FP + FN + TN}$$

Error

$$\frac{FP + FN}{TP + FP + FN + TN}$$

Precision

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

Recall

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

F1 score

$$\text{F1} = 2 \times \frac{Prec \times Rec}{Prec + Rec}$$

Prediction

Actual value	**pos**	**neg**	**Tot.**
pos′	True positive (TP)	False negative (FN)	TP + FN
neg′	False positive (FP)	True negative (TN)	FP + TN
Tot.	TP + FP	FN + TN	

DataScience Hub

Performance – error

Accuracy

$$\frac{TP + TN}{TP + FP + FN + TN}$$

Error

$$\frac{FP + FN}{TP + FP + FN + TN}$$

Precision

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

Recall

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

F1 score

$$\text{F1} = 2 \times \frac{Prec \times Rec}{Prec + Rec}$$

Prediction

Actual value	**pos**	**neg**	**Tot.**
pos′	True positive (TP)	False negative (FN)	TP + FN
neg′	False positive (FP)	True negative (TN)	FP + TN
Tot.	TP + FP	FN + TN	

Performance – precision

Accuracy

$$\frac{TP + TN}{TP + FP + FN + TN}$$

Error

$$\frac{FP + FN}{TP + FP + FN + TN}$$

Precision

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

Recall

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

F1 score

$$\text{F1} = 2 \times \frac{Prec \times Rec}{Prec + Rec}$$

Prediction

Actual value	**pos**	**neg**	**Tot.**
pos′	True positive (TP)	False negative (FN)	TP + FN
neg′	False positive (FP)	True negative (TN)	FP + TN
Tot.	TP + FP	FN + TN	

Performance – recall

Accuracy

$$\frac{TP + TN}{TP + FP + FN + TN}$$

Error

$$\frac{FP + FN}{TP + FP + FN + TN}$$

Precision

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

Recall

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

F1 score

$$\mathrm{F1} = 2 \times \frac{Prec \times Rec}{Prec + Rec}$$

Prediction

Actual value	**pos**	**neg**	**Tot.**
pos′	True positive (TP)	False negative (FN)	TP + FN
neg′	False positive (FP)	True negative (TN)	FP + TN
Tot.	TP + FP	FN + TN	

Performance – F1 score

Accuracy

$$\frac{TP + TN}{TP + FP + FN + TN}$$

Error

$$\frac{FP + FN}{TP + FP + FN + TN}$$

Precision

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

Recall

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

F1 score

$$\mathrm{F1} = 2 \times \frac{Prec \times Rec}{Prec + Rec}$$

Prediction

Actual value	**pos**	**neg**	**Tot.**
pos′	True positive (TP)	False negative (FN)	TP + FN
neg′	False positive (FP)	True negative (TN)	FP + TN
Tot.	TP + FP	FN + TN	

What if costs are not equal?

- The costs for the 'boxes' / categories might differ
- What if the cost of missing a sick person are much higher than missing a healthy person?
  - If 0=Healthy, 1=sick → missing sick = 2x healthy → FN == 2x FP
- precision is for the scared bureaucrat, embarrassed to get a wrong prediction
- recall is the superhero that needs to defuse all the bombs and not miss one

- See CompStat Munich clip (link). Focus on 3.12-6.10 for the explanation on using a cost function. The remainder of the clip talks about other cost functions (Brier etc.) – advanced topic
- Also be mindful of imbalanced classes (See the start of the CompStat Munich link)

Representations of risk structure matters

Question: 0.1% of the population has a disease, and a test detects it 99% of the time but falsely identifies 5% of healthy people as sick. What is the likelihood of a positive test result being accurate?
Each format below conveys the same fundamental information about risk structure.

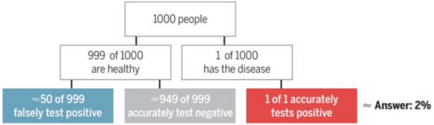
A Single-event probability format
Bayes' theorem is necessary (and difficult) when using single-event probabilities to calculate the probability of a hypothesis (having the disease) given the evidence for it (a positive test result).

P(disease) = 0.1% prevalence of disease
P(positive test|disease) = 99% true positive rate
P(positive test|no disease) = 5% false positive rate

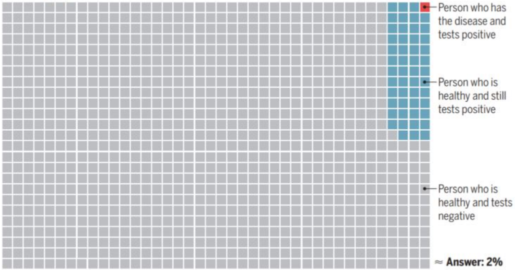
$$\text{P(disease|positive test)} = \frac{0.1\% \times 99\%}{(0.1\% \times 99\%) + (99.9\% \times 5\%)} = 1.94\% \approx \textbf{Answer: 2\%}$$

B Nested set format
This view facilitates accurate judgment because it represents base rates (prevalence) and reference class size (1 of 1000) without having to multiply a conditional probability by the base rate.

$$\text{P(disease|positive test)} = \frac{\text{P(disease AND positive test)}}{\text{P(positive test)}} = \frac{1}{50+1} = 1.96\% \approx \textbf{Answer: 2\%}$$



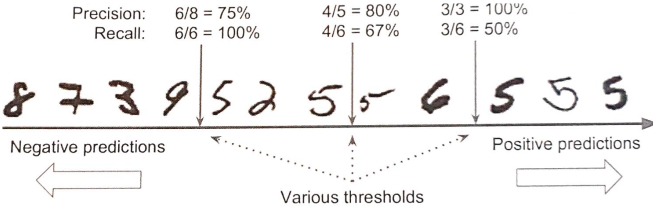
C Pictograph format

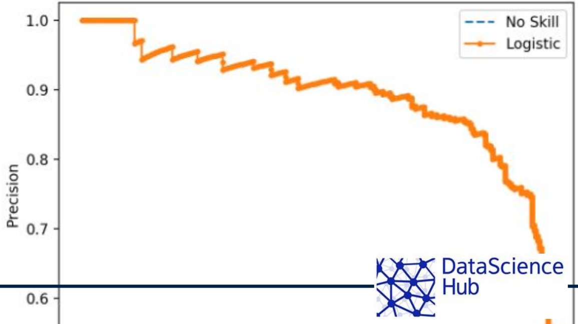


Source: Operskalski and Barbey (2016)

What about the trade-off between precision and recall

- Choosing the right threshold can lead to any level of precision
- we can see the trade-off if we plot recall against precision
- More common Receiver Operating Curve or ROC
  - True Positive Rate == Recall
    False Positive Rate == FP/(FP+TN) == 1 - Specificity





DataScience Hub

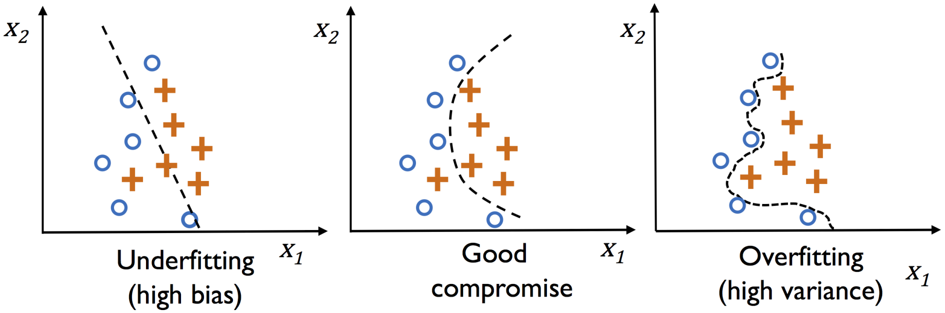
Outline

Overfitting

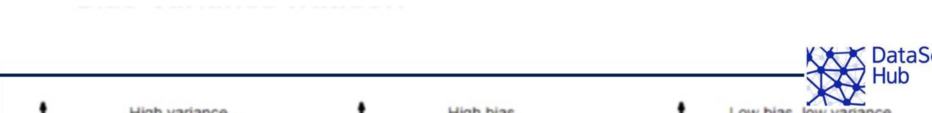
- Overfitting: sometimes model performs well on training data → low bias . . .
  . . . but does not generalize well to unseen data (test data)
  - If a model suffers from overfitting, the model has a high variance
  - This is often caused by a model that's too complex
- Underfitting can also occur (high bias)
  - Underfitting is caused by a model's not being complex enough
- Both suffer from low performance on unseen data

Bias-variance trade-off

Seen as classification

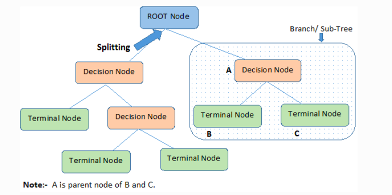


Seen as regression



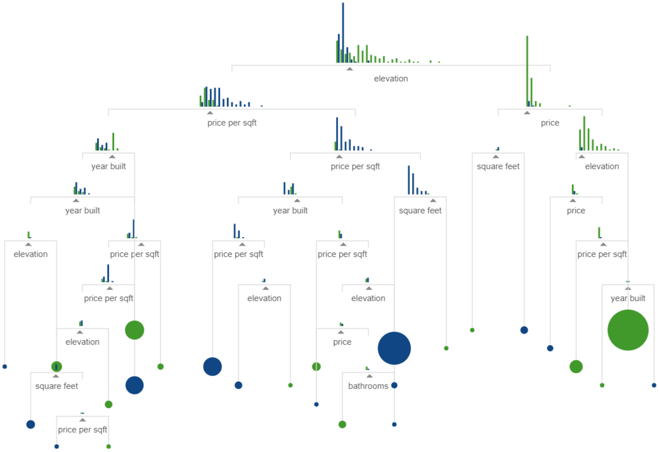
Tree jargon

- Root Node: the entire population which gets divided into two or more homogeneous sets
- Splitting: It is a process of dividing a node into two or more sub-nodes
- Decision Node: When a sub-node splits into further sub-nodes
- Leaf/Terminal Node: Nodes that do not split
- Pruning: When we remove sub-nodes of a decision node. The opposite process of splitting
- Branch/Sub-Tree: Subsection of the entire tree
- Parent/Child Node: A node, which is divided into sub-nodes is a parent node of sub-nodes. Sub-nodes are the children



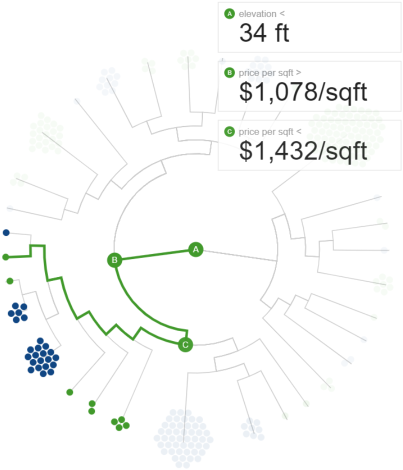
DataScience Hub

Decision tree



Source: r2d3

Bias-Variance trade-off



Source: r2d3

Outline

DataScience Hub

Regularization

- Regularization is a way to tune the complexity of the model
- Regularization helps to filter out noise from training data
- As a result, regularization prevents overfitting
- There are two main forms of regularization
    1. L1 regularization: covered in Lecture *ML3 – dimensionality and assessment*
    2. L2 regularization: weight decay

L2 regularization

The most common form of regularization is the so-called L2 regularization (sometimes also called L2 shrinkage or weight decay):

$$\frac{\lambda}{2}\|\mathbf{w}\|^2 = \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2$$

Where λ is the so-called regularization parameter. To apply regularization, we add the regularization term to the cost function, which shrinks the weights:

$$J(\mathbf{w}) = \sum_{i=1}^{n}\left[-y^{(i)}\log\left(\phi(z^{(i)})\right) - \left(1-y^{(i)}\right)\log\left(1-\phi(z^{(i)})\right)\right] + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

DataScience Hub

Regularization parameter

- We control how well we fit the training data via the regularization parameter λ
- By increasing λ, we increase the strength of regularization
- Sometimes (e.g. in scikit-learn), Support Vector Machine (SVM) terminology is used

$$C = \frac{1}{\lambda}$$

- I.e. we rewrite the regularized cost function of logistic regression as:

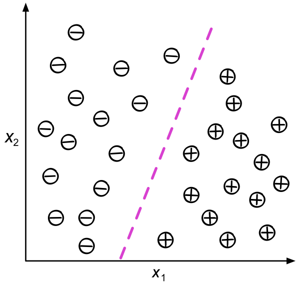
$$C\left[\sum_{i=1}^{n}\Big(-y^{(i)}\log\big(\phi(z^{(i)}\big)-\big(1-y^{(i)}\big)\Big)\log\left(1-\phi(z^{(i)})\right)\right]+\frac{1}{2}\|\mathbf{w}\|^2$$

DataScience Hub

Outline

Recap:

- Predict categorical class labels based on past observations
- Class labels are discrete unordered values which cannot be ordered.
- That is, each element of Y represents a class label and output Y consists of a discrete set of outcomes
- Examples
  - Binary: Email spam classification example or unemployment status
  - Multi-class: Handwritten digit classification example



What is a hyperplane?

- In a p-dimensional space, it's a flat affine subspace of dimension p-1.
  - Example: in a 2-dimensional space, it's a 1-dimensional line.
  - Example: in a 3-dimensional space, it's a 2-dimensional plane.

- Formal definition:
  - For a 2-dimensional hyperplane: $\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$
  - For a p-dimensional hyperplane: $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_P X_P = 0$
  - Any point that satisfies the equation lies on the hyperplane.

- Suppose that X instead satisfies

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_P X_P > 0 \quad \rightarrow \text{lies "above" the hyperplane}$$
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_P X_P < 0 \quad \rightarrow \text{lies "below" the hyperplane}$$

How to classify using a hyperplane?

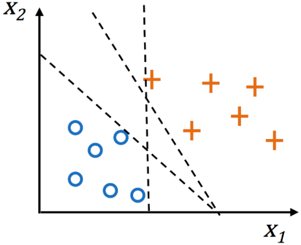
- A separating hyperplane then has the properties:

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} > 0, \text{ if } y_i = 1$$

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} < 0, \text{ if } y_i = -1$$

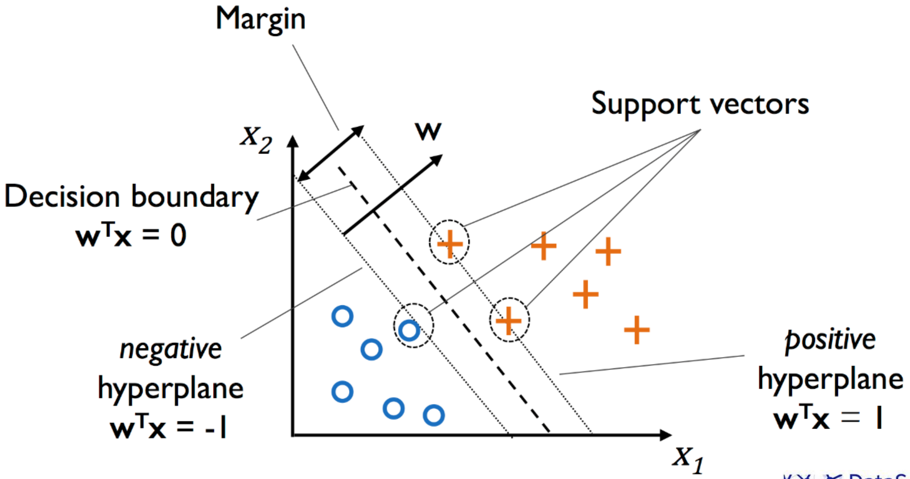
- Or, put differently:

$$y_i \left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}\right) > 0 \text{ for all } i$$



- This gives an infinite number of planes
- But which plane is the optimal one?
  $\rightarrow$ maximum margin classification

Maximum margin classification



See “Decision Boundary” (Andrew Ng) in Knowledge clips

Support Vector Machines (SVM)

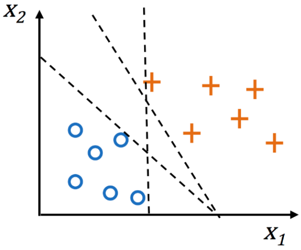
- In SVMs, the optimization objective is to maximize the margin
- The margin is defined as the distance between the separating hyperplane and the training samples that are closest to this hyperplane (support vectors)
- Intuitively, the larger the margin, the lower generalization error (variance)
- Models with small margin are prone to overfitting

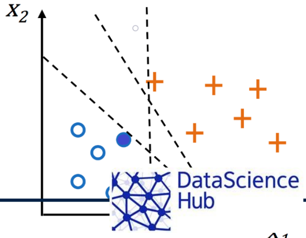
See "How do Support Vector Machines work?" (Brandon Rohrer) in Knowledge clips.

SVM: 2 undesirable properties

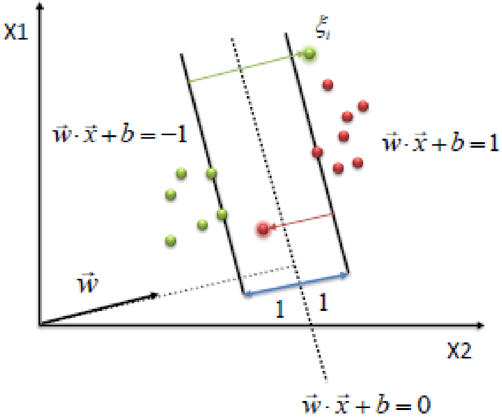
1. It is very sensitive to small changes in the training set, especially the supports
   - What if the blue dot is removed?
2. It assumes that a separating hyperplane exists

Solution: introduce a slack variable ξ_i that allows some instances to 'cross' the margin but then penalize this





Slack variables / soft margin SVM



slack variable:

ξ_i

Allow some instances to fall off the margin, but penalize them

Extending SVM to non-linearly separable cases

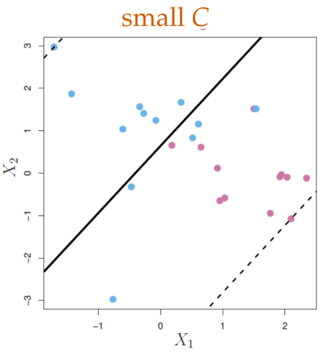
- To ensure convergence in presence of misclassifications we need to relax the linear constraints
- Introduce slack variables ξ

$$\mathbf{w}^T\mathbf{x}^{(i)} \geq 1 - \xi^{(i)} \text{ if } y^{(i)} = 1$$

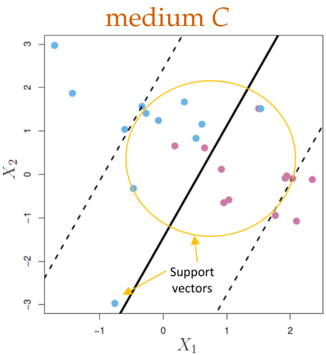
$$\mathbf{w}^T\mathbf{x}^{(i)} < -1 + \xi^{(i)} \text{ if } y^{(i)} = -1$$

- New objective to be minimized: $\frac{1}{2}\|\mathbf{w}\|^2 + C\left(\sum_i \xi^{(i)}\right)$
- C controls width of the margin with large values of $C \equiv$ large error penalties
- C is a way to do regularization in SVMs

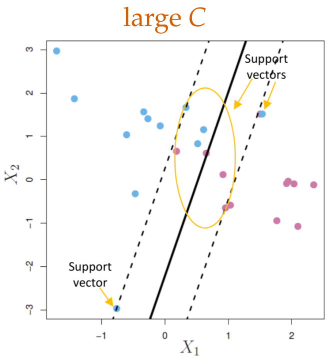
What is the effect of different values of C?



- All observations are support vectors
- High bias, low variance



- Most observations are support vectors here



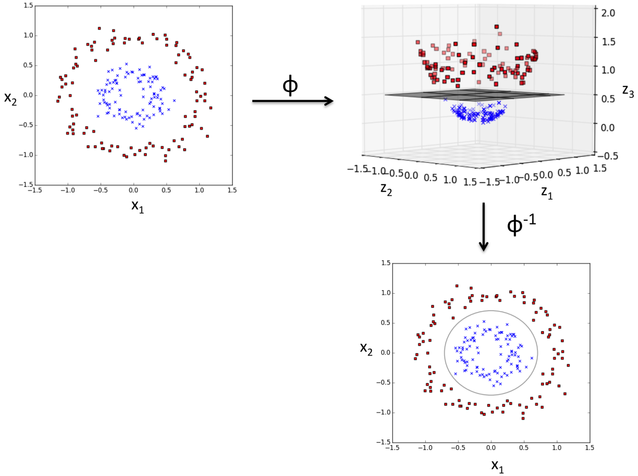
- Few support vectors
- Low bias, high variance

DataScience Hub

Non-linear relations can also cause non-separability

- With non-linear relationships, a support vector classifier is problematic
- One way to add non-linearity is to add higher order terms $(x_i^2, x_i^3, \ldots)$ into the separating hyperplane, but this is often computationally impractical.
- A popular alternative is using a kernel
  - linear kernel $K(x_i, x_{i'}) = \sum_{i=1}^{p} x_{ij}x_{i'j}$
  - polynomial kernel $K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij}x_{i'j}\right)^d$
  - radial kernel $K(x_i, x_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2\right)$
- General blueprint:
  - Transform training data into a higher dimensional space via a mapping function $\phi(\cdot)$
  - Train a linear SVM to classify the data in the new feature space
  - Use the same mapping function $\phi(\cdot)$ to transform new (unseen) data
  - Classify unseen data using the linear SVM model

Applying a radial kernel



How to use SVM with more than two classes?

- A separating hyperplane approach does not work for multiple classes

Workarounds:

1. One-versus-One classification approach
   - Take all possible pairs, compare them one-versus-one, and create several classifiers. e.g., if 3 classes (A,B,C), then compare A with B, A with C, and B with C
   - Then, for a given test observation, run it through all *K* classifiers and tally the number of times it has be assigned to any of the classes
   - Assign it the class to which it was predicted to belong the most often
2. One-versus-All classification approach
   - Fit *K* SVMs, each time comparing one of the *K* classes to the remaining *K*-1 classes (treated as if they were one combined class)
   - For a given test observation, run it through all *K* classifiers and assign it to the class for which it was most frequently predicted

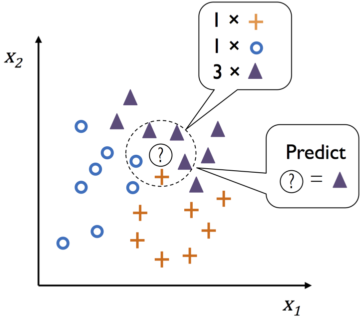
Outline

ML2 – the basics

- Measuring success
- Overfitting, bias and variance
- Fixing overfitting
- Classification: Support Vector Machines (SVM)
- K-nearest neighbors (KNN)

K-nearest neighbors (KNN)

- KNN is an example of a non-parametric model
  - Parametric models learn parameters from training data
  - Once training done, the training set not required
- KNN is an instance-based or lazy learner
  - so needs all the data, all the time
- Basic KNN algorithm
  1. Choose *k* and a distance metric
  2. Find *k* nearest neighbors of the sample to be classified
  3. Assign the class label by majority vote



KNN – continued

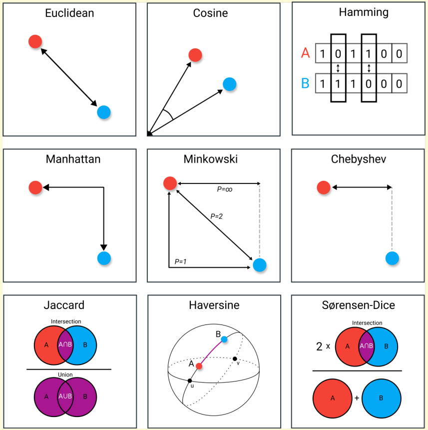
- Distance metrics:

$$d\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \sqrt[p]{\sum_k \left|x_k^{(i)} - x_k^{(j)}\right|^p}$$

  Euclidean distance if we set the parameter $p = 2$
  Manhattan distance if we set the parameter $p = 1$
  (cf. Hamming distance and Minkowski distance)

- Classifier immediately adapts as we receive new training examples . . .
- . . . so computational complexity grows linearly with the number of samples
- Need for efficient data structures such as KD-trees

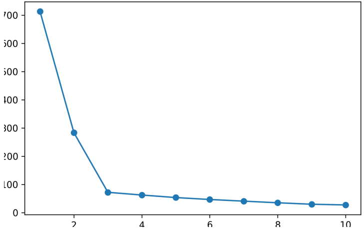
What is the right distance?



Source: Maarten Grootendorst on Medium

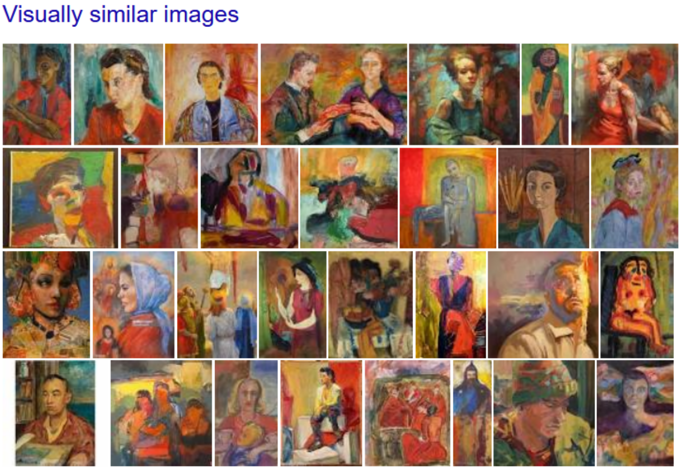
How to choose the 'right' number of neighbors?

- As you've seen before, there is no easy way to answer this.
- Increasing *k* will lead to high bias but low variance etc. etc.
- Playing around with *k* will show you the best performance in your model
- Maybe an elbow graph can be of use
  - Pick the point where increasing *k* does not improve the model much
  - See a similar discussion for *K*-means



Number of neighbors

KNN – applications in image recognition



Source: Labnol.org

KNN – applications in finance

- Credit ratings, financial institutes will predict the credit rating of customers
- In loan disbursement, banks predict whether the loan is safe or risky.
- In a regression framework: KNN can be used to predict financial time series
  - stock markets, FX rates, etc., etc.

Summary

In this lecture we covered:

1. The balance between bias and variance
2. Looked at various measures of success in classification
   - Confusion matrix, Accuracy, Precision, Recall
3. Discussed how Support Vector Machines and K Nearest Neighbours work

Deriving the partial derivative: step-by-step

$$
\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_i \left(y^{(i)} - \phi(z^{(i)})\right)^2 \\
&= \frac{1}{2}\frac{\partial}{\partial w_j}\sum_i \left(y^{(i)} - \phi(z^{(i)})\right)^2 \\
&= \frac{1}{2}\sum_i 2\big(y^{(i)} - \phi(z^{(i)})\big)\frac{\partial}{\partial w_j}\left(y^{(i)} - \phi(z^{(i)})\right) \\
&= \sum_i \big(y^{(i)} - \phi(z^{(i)})\big)\frac{\partial}{\partial w_j}\left(y^{(i)} - \sum_i \big(w_j^{(i)} x_j^{(i)}\big)\right) \\
&= \sum_i \left(y^{(i)} - \phi(z^{(i)})\right)\left(-x_j^{(i)}\right) \\
&= -\sum_i \left(y^{(i)} - \phi(z^{(i)})\right)x_j^{(i)}
\end{aligned}
$$

DataScience Hub

Weight update derivation

Calculate the partial derivative of the log-likelihood function with respect to the jth weight:

$$\frac{\partial}{\partial w_j} l(\mathbf{w}) = \left(y \frac{1}{\phi(z)} - (1-y) \frac{1}{1-\phi(z)} \right) \frac{\partial}{\partial w_j} \phi(z)$$

Partial derivative of the sigmoid function:

$$\frac{\partial}{\partial z} \phi(z) = \frac{\partial}{\partial z} \frac{1}{1+e^{-z}} = \frac{1}{\left(1+e^{-z}\right)^2} e^{-z} = \frac{1}{1+e^{-z}} \left(1 - \frac{1}{1+e^{-z}} \right)$$

$$= \phi(z)(1-\phi(z)).$$

DataScience Hub

Weight update derivation

Resubstitute $\frac{\partial}{\partial z}\phi(z) = \phi(z)(1-\phi(z))$ to obtain:

$$
\begin{aligned}
&\left(y\frac{1}{\phi(z)} - (1-y)\frac{1}{1-\phi(z)}\right)\frac{\partial}{\partial w_j}\phi(z) \\
&= \left(y\frac{1}{\phi(z)} - (1-y)\frac{1}{1-\phi(z)}\right)\phi(z)\big(1-\phi(z)\big)\frac{\partial}{\partial w_j}z \\
&= \Big(y\big(1-\phi(z)\big) - (1-y)\phi(z)\Big)x_j \\
&= \big(y-\phi(z)\big)x_j
\end{aligned}
$$

References I

Operskalski, J. T., & Barbey, A. K. (2016). Risk literacy in medical decision-making. Science, 352(6284), 413–414.