Applied Machine Learning

Supervised machine learning (Part 2)

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Applied Machine Learning

Naïve Bayes Classifiers

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Naïve Bayes Classifiers: a simple, probabilistic classifier family

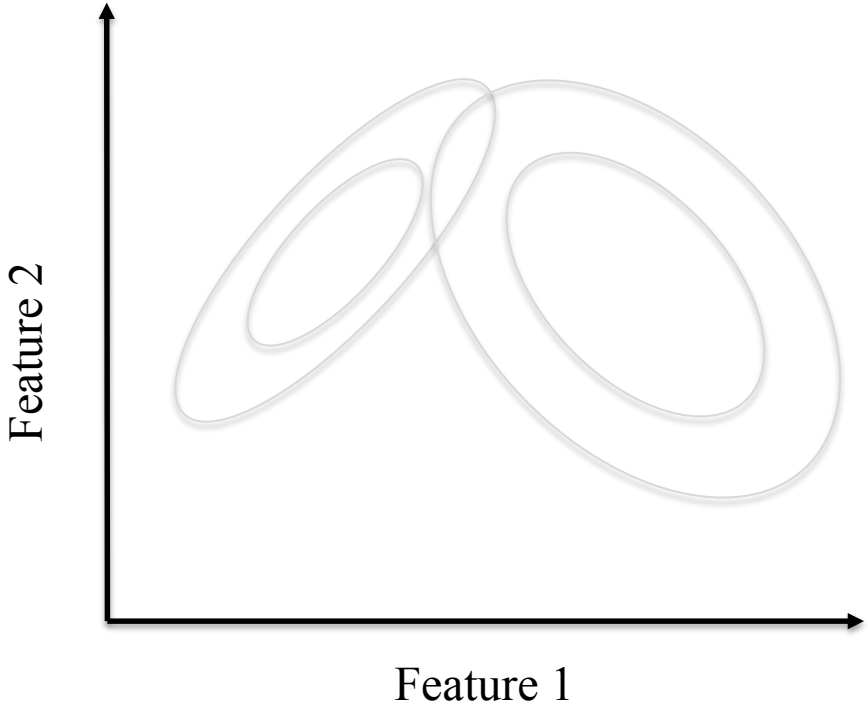
- These classifiers are called 'Naïve' because they assume that features are conditionally independent, given the class.
- In other words: they assume that, for all instances of a given class, the features have little/no correlation with each other.
- Highly efficient learning and prediction.
- But generalization performance may worse than more sophisticated learning methods.
- Can be competitive for some tasks.

Naïve Bayes classifier types

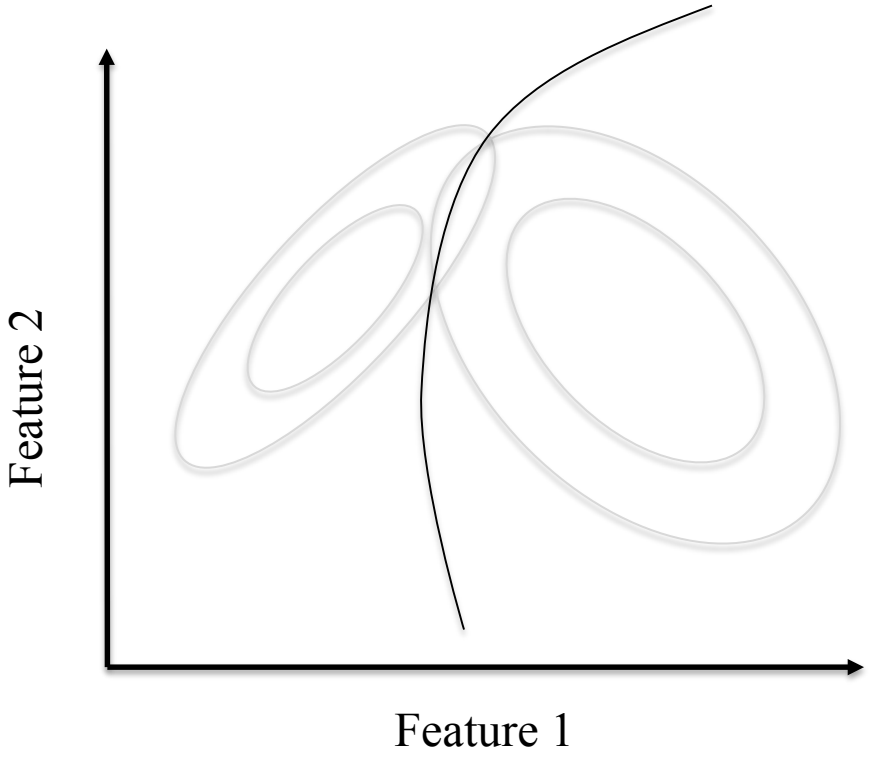
- **Bernoulli: binary features (e.g. word presence/absence)**
- **Multinomial: discrete features (e.g. word counts)**
- **Gaussian: continuous/real-valued features**
  - *Statistics computed for each class:*
    - *For each feature: mean, standard deviation*

- **See the Applied Text Mining course for more details on the Bernoulli and Multinomial Naïve Bayes models**

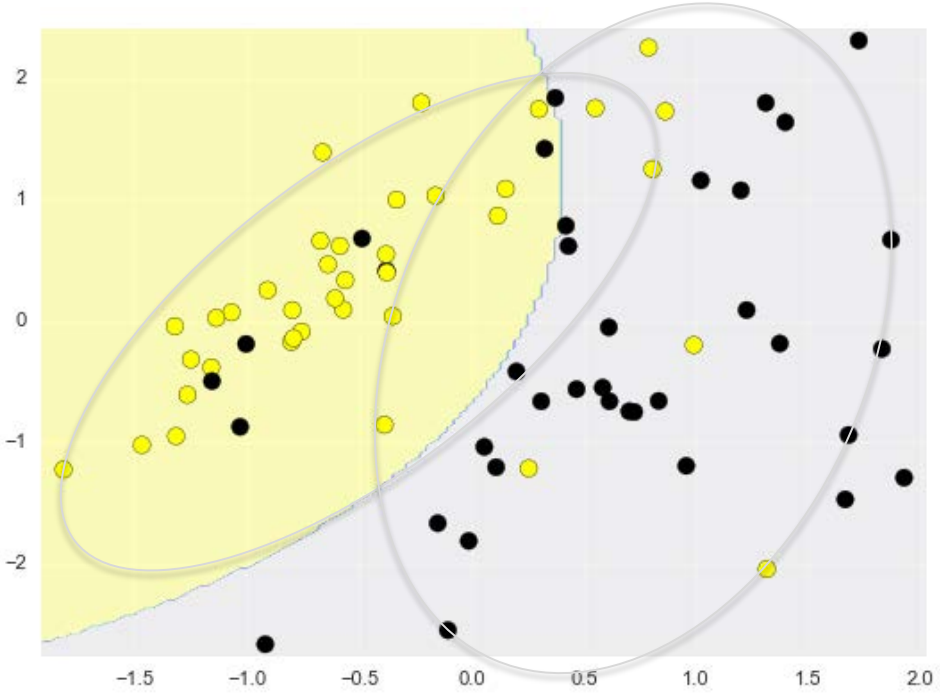
Gaussian Naïve Bayes classifier



Gaussian Naïve Bayes classifier



Gaussian Naïve Bayes classifier



Naïve Bayes classifiers: Pros and Cons

Pros:

- Easy to understand
- Simple, efficient parameter estimation
- Works well with high-dimensional data
- Often useful as a baseline comparison against more sophisticated methods

Cons:

- Assumption that features are conditionally independent given the class is not realistic.
- As a result, other classifier types often have better generalization performance.
- Their confidence estimates for predictions are not very accurate.

Applied Machine Learning

Random Forests

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Random Forests

- **An ensemble of trees, not just one tree.**
- **Widely used, very good results on many problems.**
- **`sklearn.ensemble` module:**
  - *Classification:* `RandomForestClassifier`
  - *Regression:* `RandomForestRegressor`
- **One decision tree → Prone to overfitting.**
- **Many decision trees → More stable, better generalization**
- **Ensemble of trees should be diverse: introduce random variation into tree-building.**

Random Forest Process

Original dataset

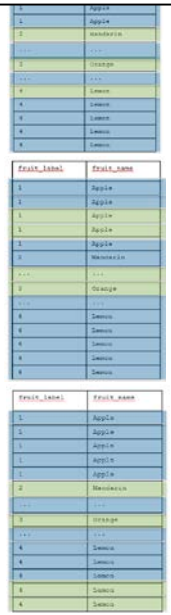
Randomized bootstrap copies

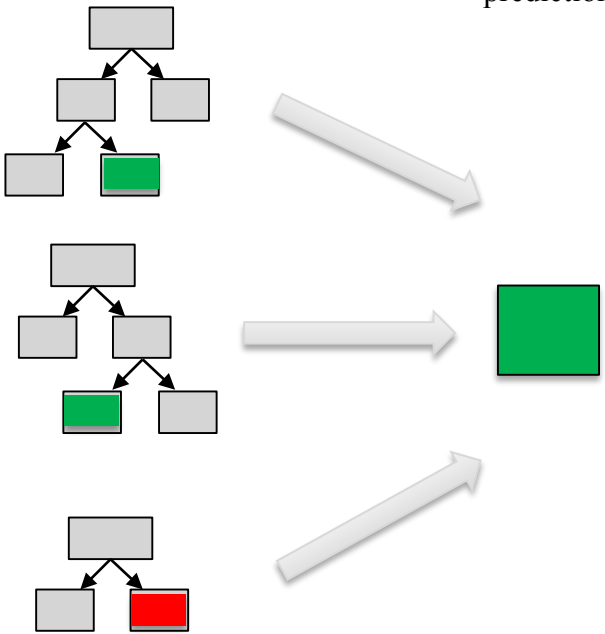
n_estimator

Randomized feature splits

Ensemble prediction

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon





Random Forest Process: Bootstrap Samples

Bootstrap sample 1

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

Bootstrap sample 2

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

Bootstrap sample 3

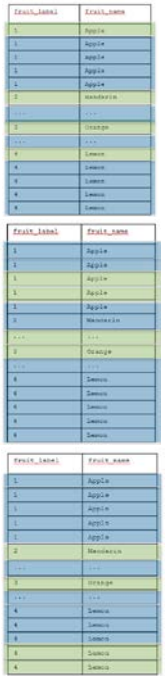
fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

Random Forest Process

Original dataset

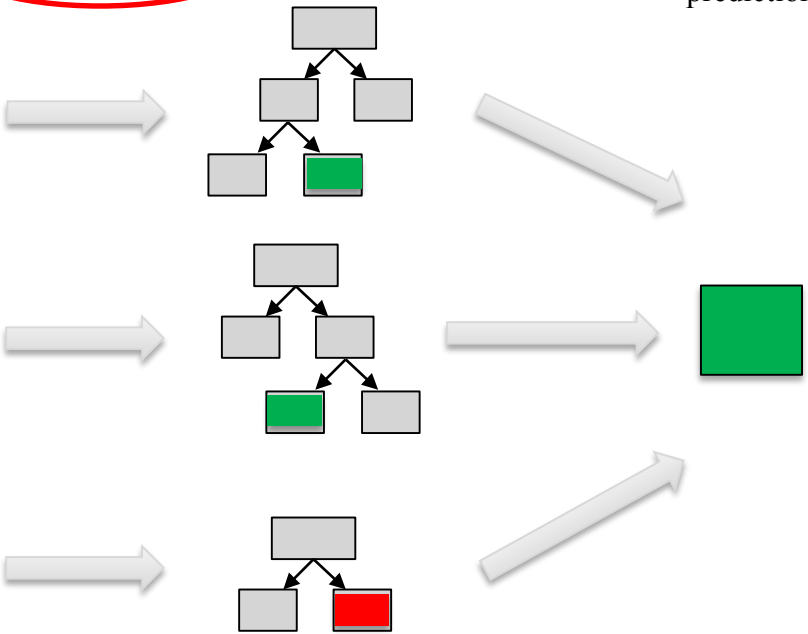
fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

Randomized bootstrap copies



Randomized feature splits

max_features



Ensemble prediction

Random Forest `max_features` Parameter

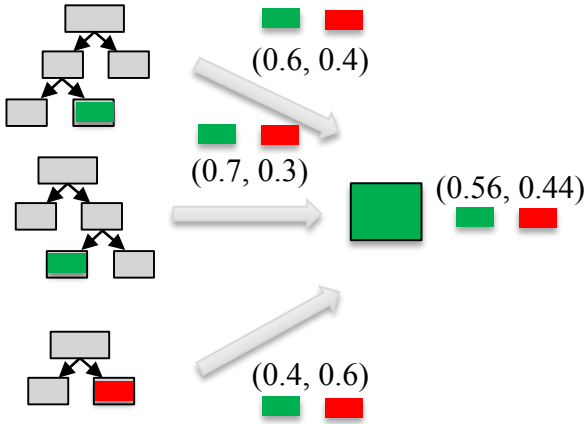
- Learning is quite sensitive to `max_features`.
- Setting `max_features = 1` leads to forests with diverse, more complex trees.
- Setting `max_features = <close to number of features>` will lead to similar forests with simpler trees.

Prediction Using Random Forests

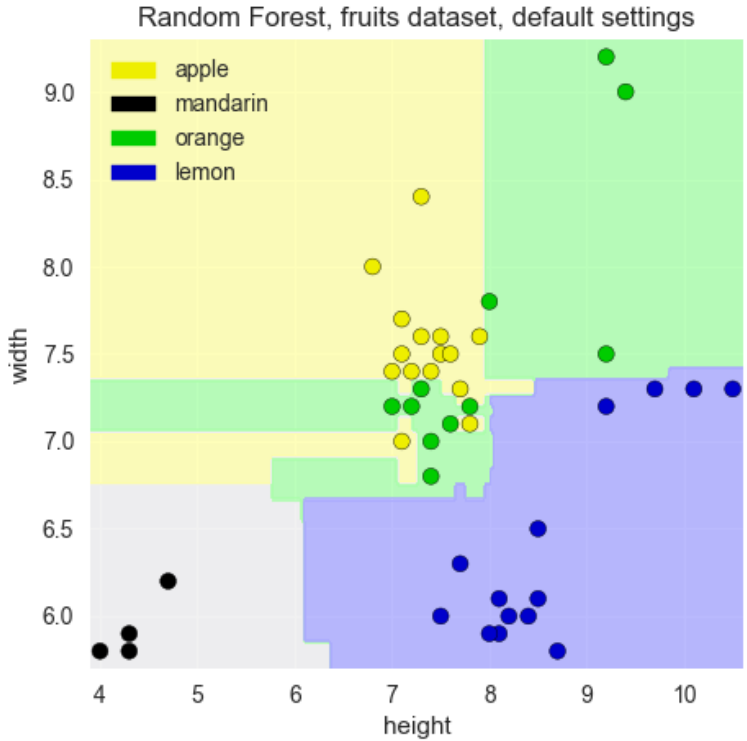
1. Make a prediction for every tree in the forest.

2. Combine individual predictions

- *Regression: mean of individual tree predictions.*
- *Classification:*
  - *Each tree gives probability for each class.*
  - *Probabilities averaged across trees.*
  - *Predict the class with highest probability.*



Random Forest: Fruit Dataset



Random Forest: Pros and Cons

Pros:

- Widely used, excellent prediction performance on many problems.
- Doesn't require careful normalization of features or extensive parameter tuning.
- Like decision trees, handles a mixture of feature types.
- Easily parallelized across multiple CPUs.

Cons:

- The resulting models are often difficult for humans to interpret.
- Like decision trees, random forests may not be a good choice for very high-dimensional tasks (e.g. text classifiers) compared to fast, accurate linear models.

Random Forests: RandomForestClassifier Key Parameters

- `n_estimators`: number of trees to use in ensemble (default: 10).
  - *Should be larger for larger datasets to reduce overfitting (but uses more computation).*
- `max_features`: has a strong effect on performance. Influences the diversity of trees in the forest.
  - *Default works well in practice, but adjusting may lead to some further gains.*
- `max_depth`: controls the depth of each tree (default: None. Splits until all leaves are pure).
- `n_jobs`: How many cores to use in parallel during training.

- Choose a fixed setting for the random_state parameter if you need reproducible results.

Applied Machine Learning

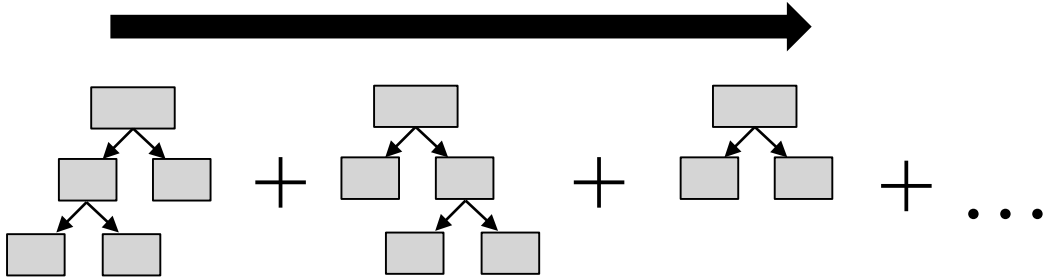
Gradient Boosted Decision Trees

Kevyn Collins-Thompson

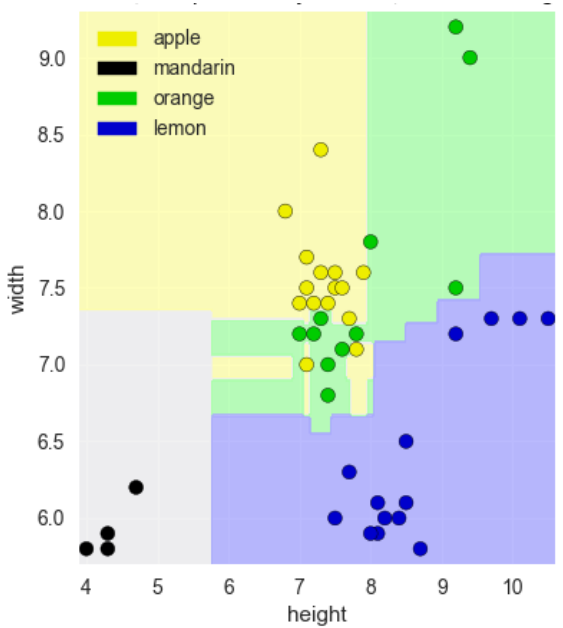
Associate Professor of Information & Computer Science
University of Michigan

Gradient Boosted Decision Trees (GBDT)

- Training builds a series of small decision trees.
- Each tree attempts to correct errors from the previous stage.



- The learning rate controls how hard each new tree tries to correct remaining mistakes from previous round.
  - High learning rate: more complex trees
  - Low learning rate: simpler trees



GBDT decision regions on two-feature fruits dataset

GBDT: Pros and Cons

Pros:

- Often best off-the-shelf accuracy on many problems.
- Using model for prediction requires only modest memory and is fast.
- Doesn't require careful normalization of features to perform well.
- Like decision trees, handles a mixture of feature types.

Cons:

- Like random forests, the models are often difficult for humans to interpret.
- Requires careful tuning of the learning rate and other parameters.
- Training can require significant computation.
- Like decision trees, not recommended for text classification and other problems with very high dimensional sparse features, for accuracy and computational cost reasons.

GBDT: GradientBoostingClassifier

Key Parameters

- `n_estimators`: sets # of small decision trees to use (weak learners) in the ensemble.
- `learning_rate`: controls emphasis on fixing errors from previous iteration.
- The above two are typically tuned together.
- `n_estimators` is adjusted first, to best exploit memory and CPUs during training, then other parameters.
- `max_depth` is typically set to a small value (e.g. 3-5) for most applications.

Applied Machine Learning

Neural networks

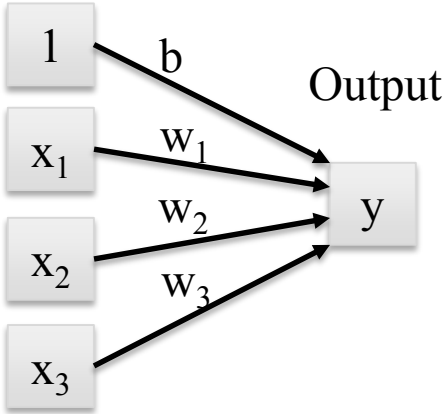
Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Review: Linear and Logistic Regression

Linear regression

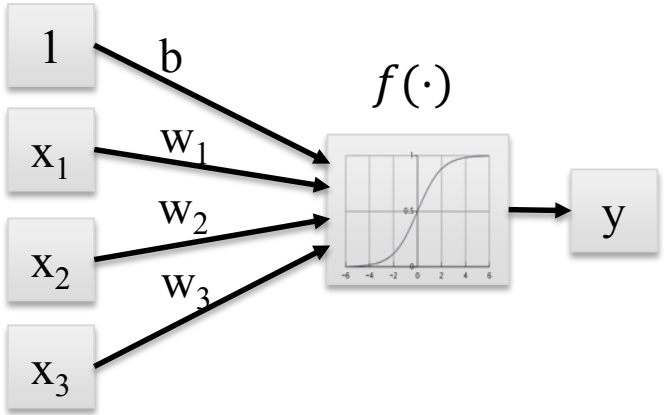
Input features



$$\hat{y} = \hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n$$

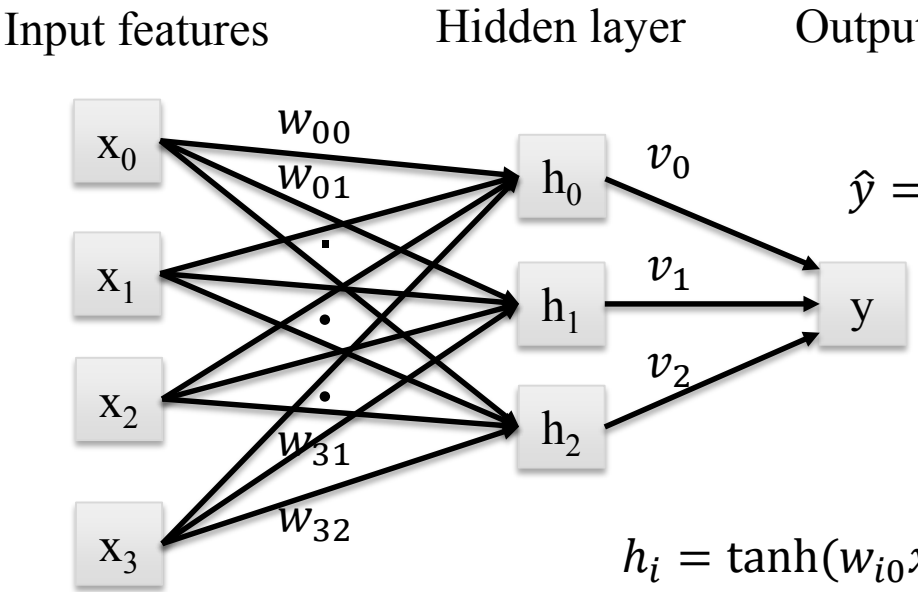
Logistic regression

Input features



$$\hat{y} = \text{logistic}(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)$$

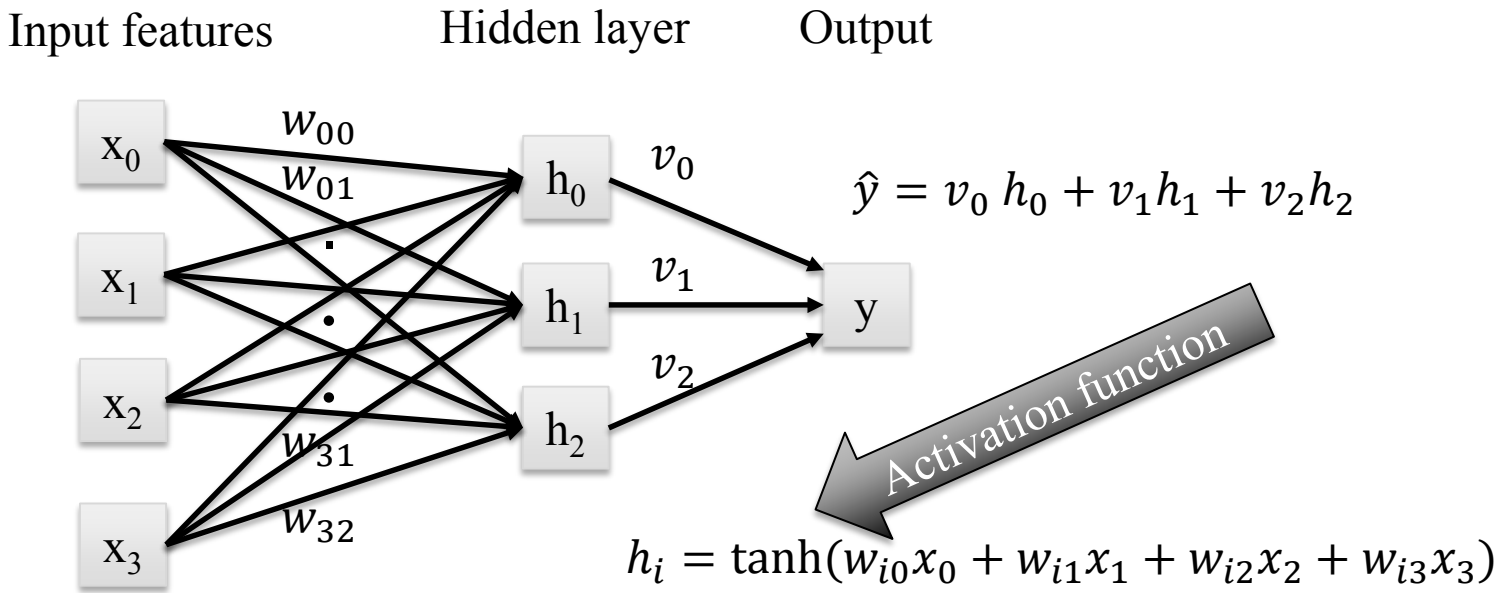
Multi-layer Perceptron with One Hidden Layer (and tanh activation function)



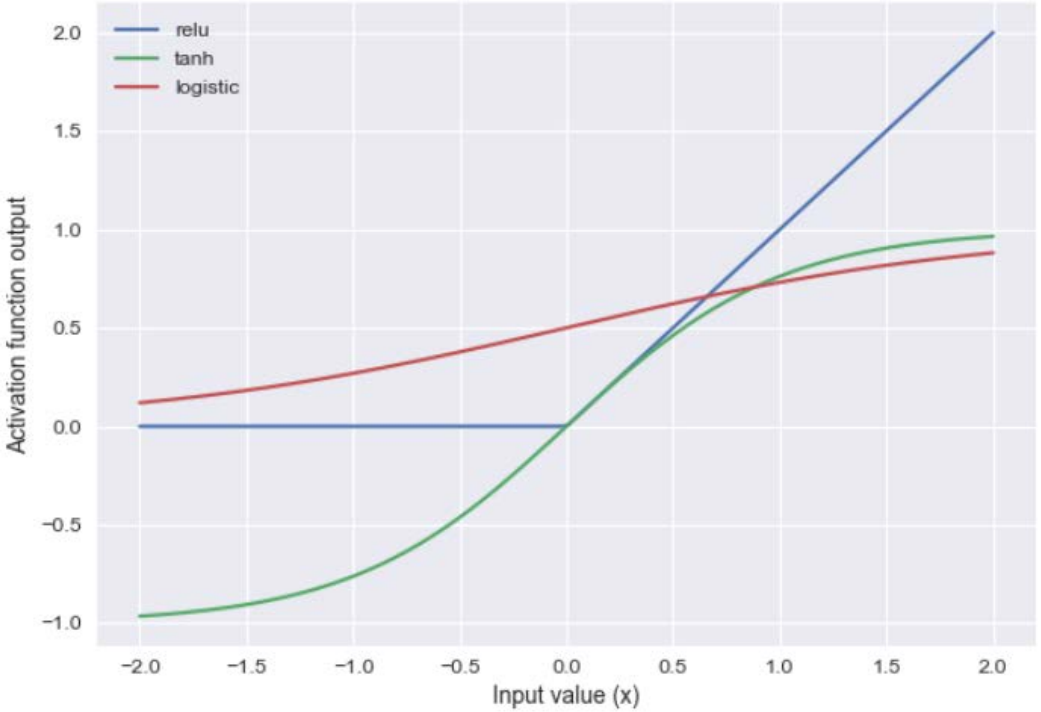
$$\hat{y} = v_0\, h_0 + v_1 h_1 + v_2 h_2$$

$$h_i = \tanh(w_{i0}x_0 + w_{i1}x_1 + w_{i2}x_2 + w_{i3}x_3)$$

Multi-layer Perceptron with One Hidden Layer (and tanh activation function)

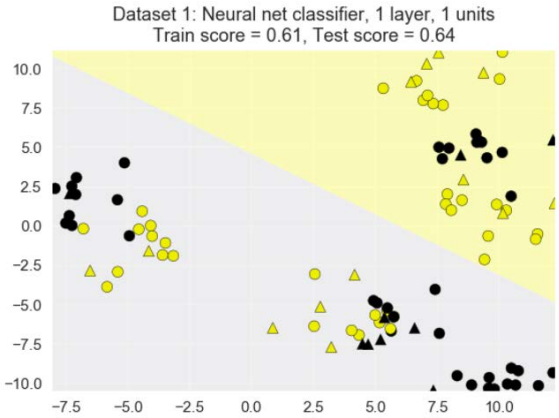


Activation Functions

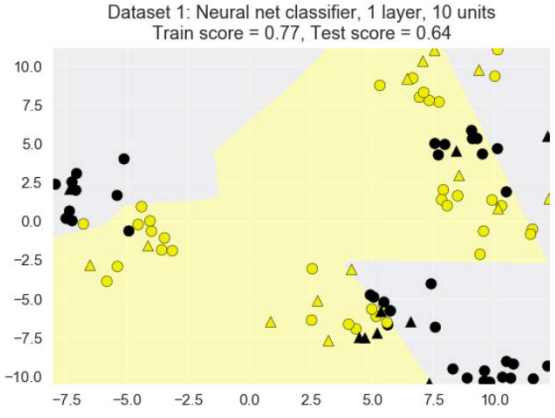


$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

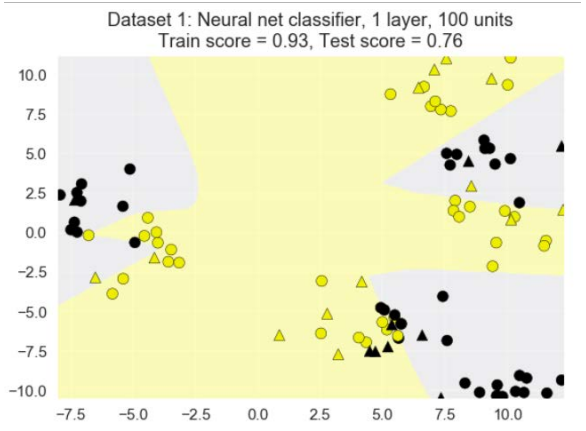
A single hidden layer network using 1, 10, or 100 units



1 layer, 1 unit

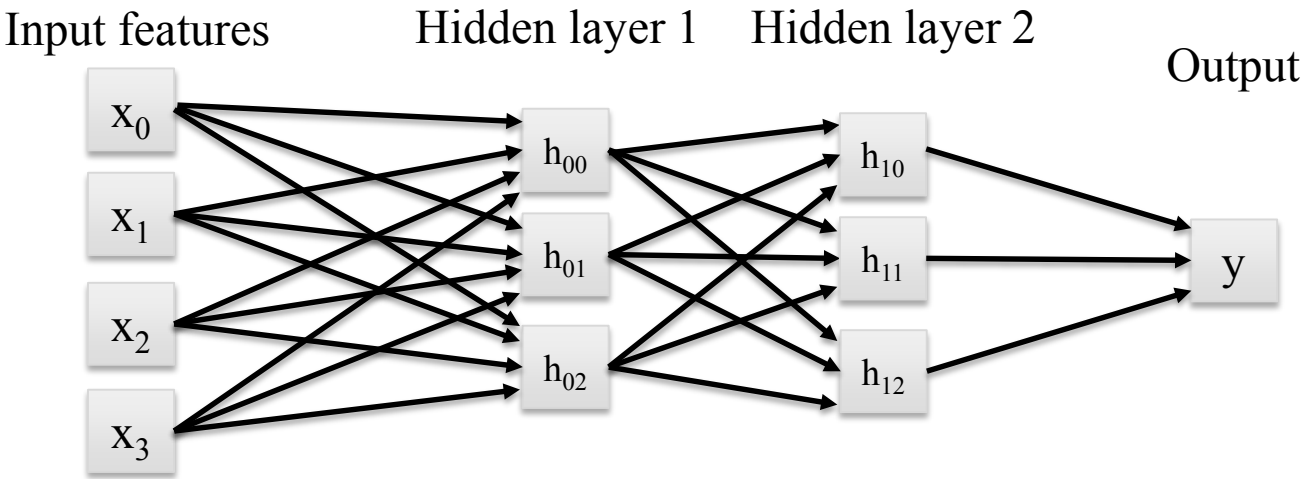


1 layer, 10 units

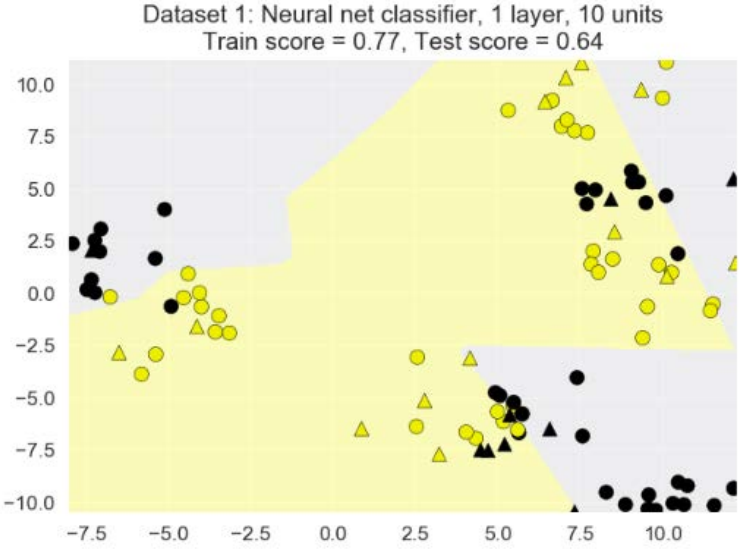


1 layer, 100 units

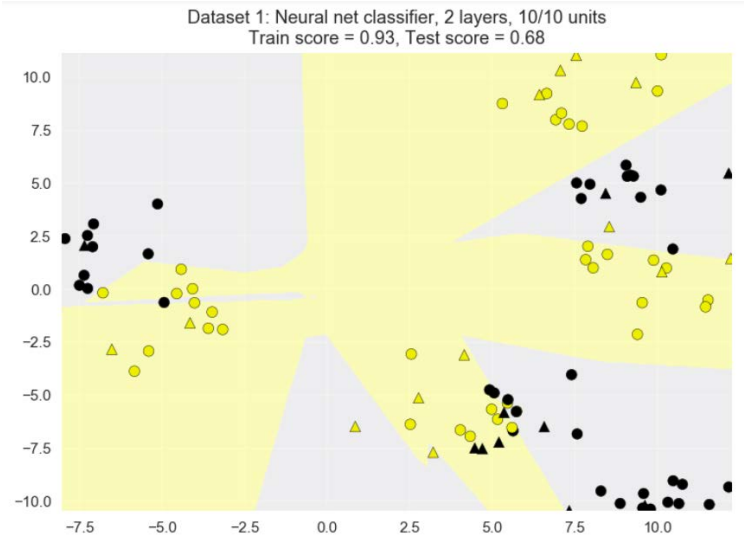
Multi-layer Perceptron with Two Hidden Layers



One vs Two Hidden Layers

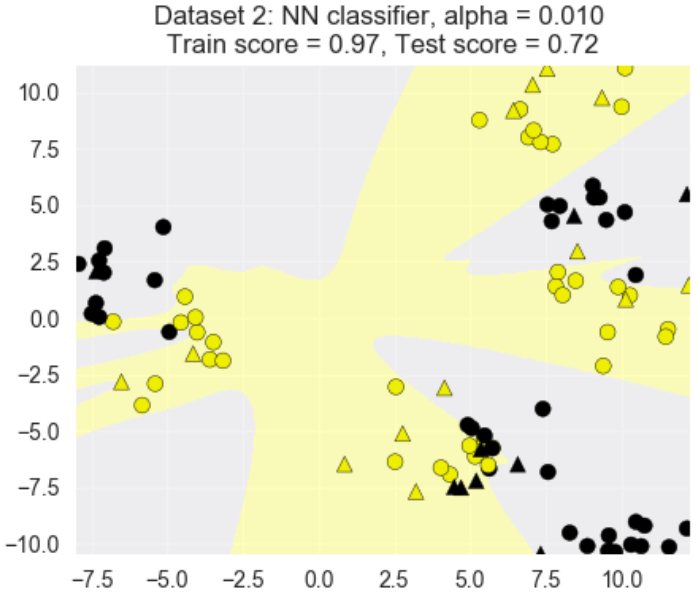


1 layer, 10 units

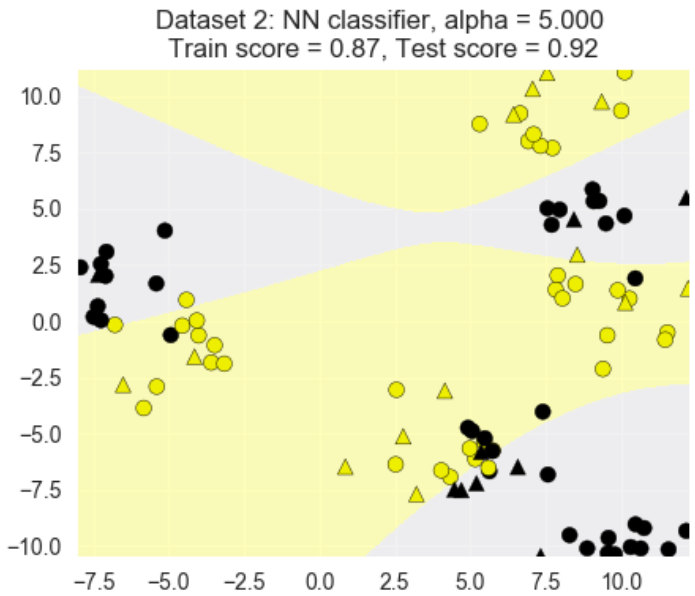


2 layers, (10, 10) units

L2 Regularization with the Alpha Parameter



alpha = 0.01

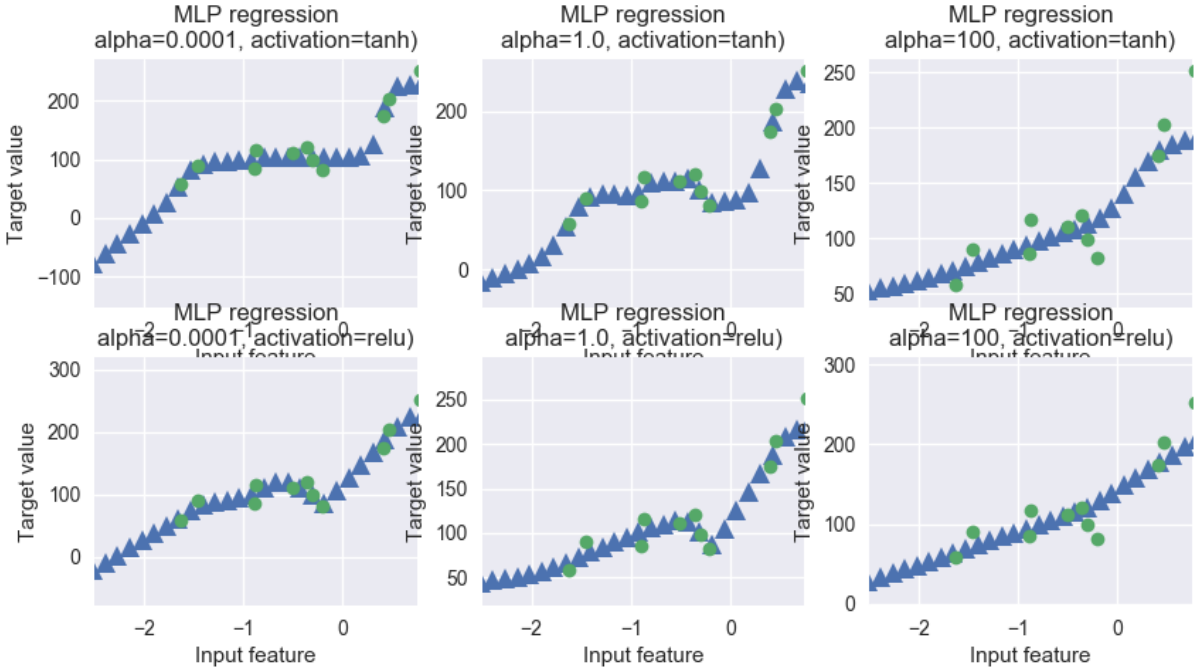


alpha = 5.0

Neural Network Regression with MLPRegressor

activation = 'tanh'

activation = 'relu'



alpha = 0.0001 alpha = 1.0 alpha = 100.0

Increasing regularization

Neural Networks: Pros and Cons

Pros:

- They form the basis of state-of-the-art models and can be formed into advanced architectures that effectively capture complex features given enough data and computation.

Cons:

- Larger, more complex models require significant training time, data, and customization.
- Careful preprocessing of the data is needed.
- A good choice when the features are of similar types, but less so when features of very different types.

Neural Nets: MLPClassifier and MLPRegressor Important pParameters

- **`hidden_layer_sizes`: sets the number of hidden layers (number of elements in list), and number of hidden units per layer (each list element).** *Default: (100).*
- **`alpha`: controls weight on the regularization penalty that shrinks weights to zero.** *Default: alpha = 0.0001.*
- **`activation`: controls the nonlinear function used for the activation function, including: `'relu'` (default), `'logistic'`, `'tanh'`.**

Applied Machine Learning

Deep Learning
(Optional overview)

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Deep Learning Summary

- Deep learning architectures combine a sophisticated automatic feature extraction phase with a supervised learning phase.
- The feature extraction phase uses a hierarchy of multiple feature extraction layers.
- Starting from primitive, low-level features in the initial layer, each feature layer's output provides the input features to the next higher feature layer.
- All features are used in the final supervised learning model.

An example of a simple deep learning architecture

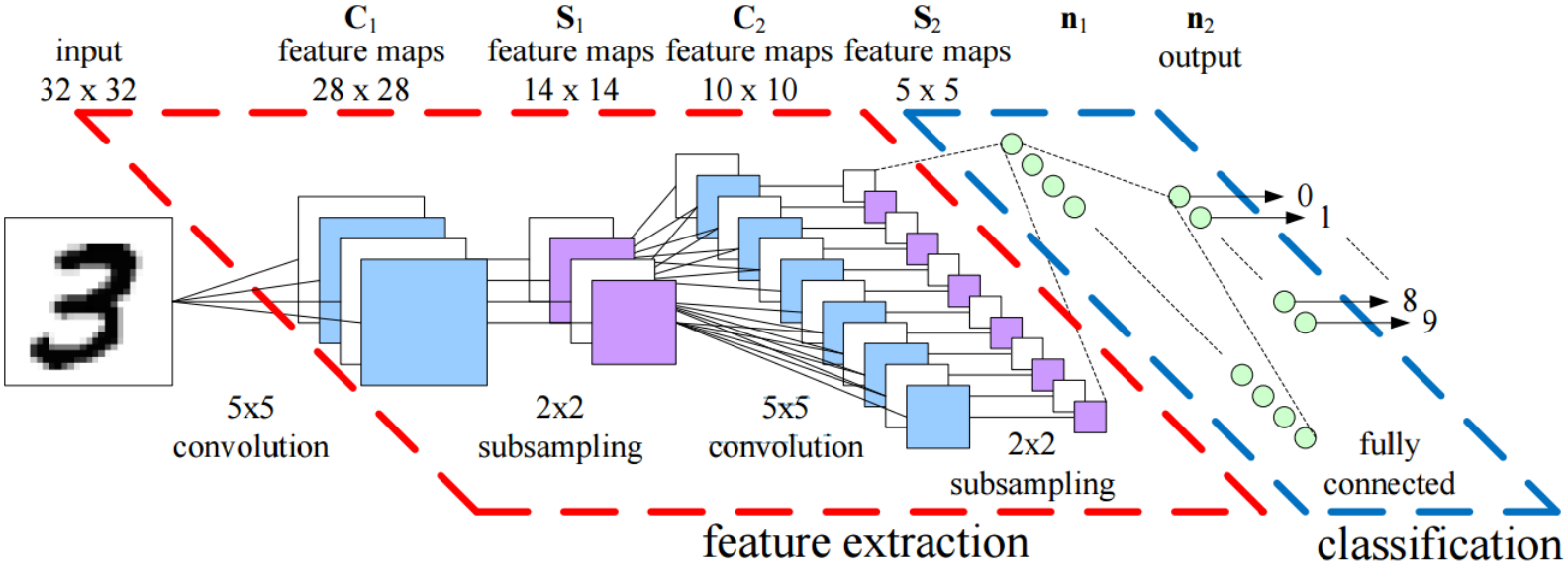
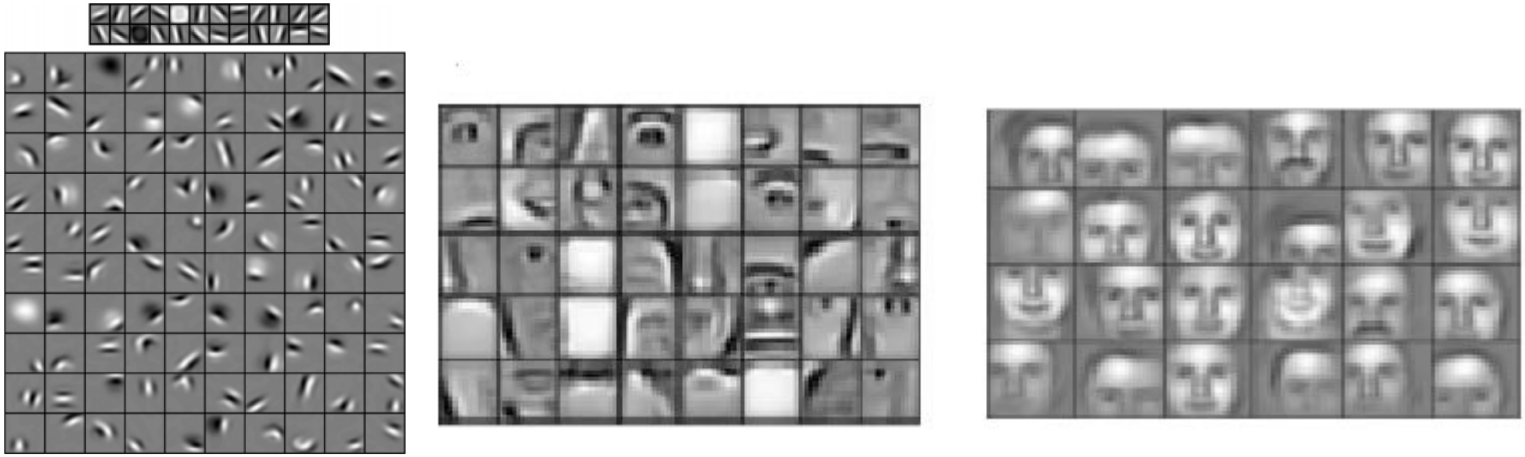


Image: M. Peemen, B. Mesman, and H. Corporaal. Efficiency Optimization of Trainable Feature Extractors for a Consumer Platform. Proceedings of the 13th International Conference on Advanced Concepts for Intelligent Vision Systems, 2011.

Deep Learning



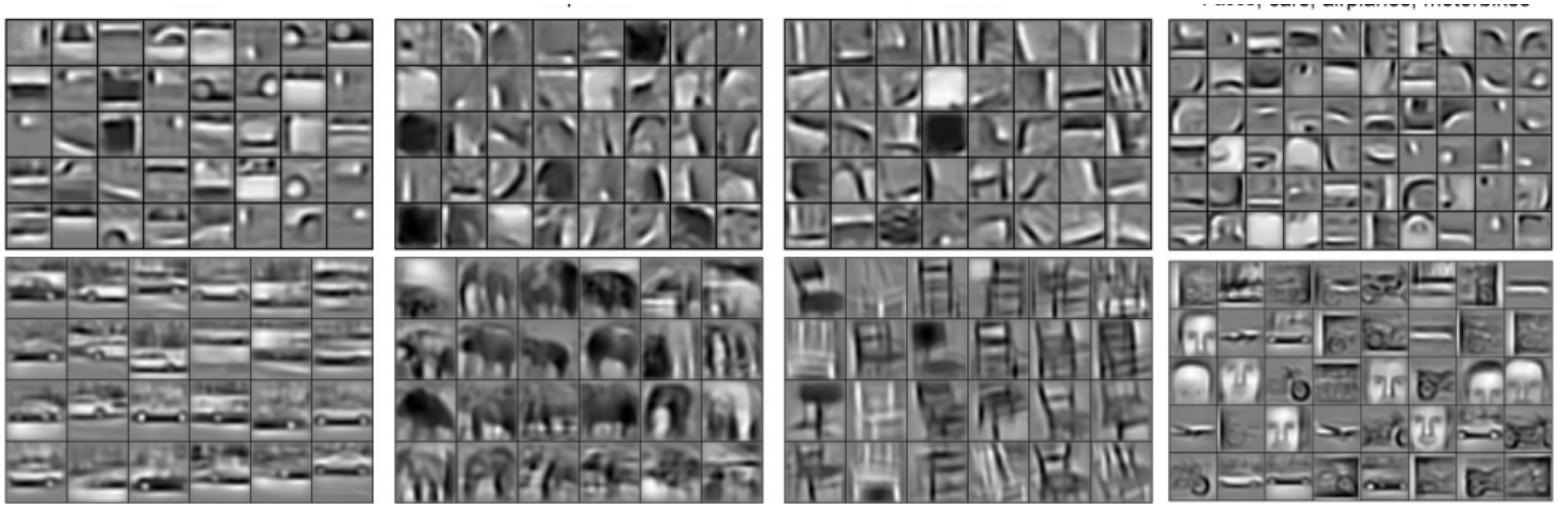
First, second, and third feature layer bases learned for faces

Image: Honglak Lee and colleagues (2011) from "Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks". Communications of the ACM, Vol. 54 No. 10, Pages 95-103

Deep Learning

Second layer

Third layer



Cars | Elephants | Chairs | Faces, cars, airplanes, motorbikes

Image: Honglak Lee and colleagues (2011) from "Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks". Communications of the ACM, Vol. 54 No. 10, Pages 95-103

Pros and Cons of Deep Learning

- **Pros:**
  - *Powerful: deep learning has achieved significant gains over other machine learning approaches on many difficult learning tasks, leading to state-of-the-art performance across many different domains.*
  - *Does effective automatic feature extraction, reducing the need for guesswork and heuristics on this key problem.*
  - *Current software provides flexible architectures that can be adapted for new domains fairly easily.*
- **Cons:**
  - *Can require huge amounts of training data.*
  - *Can require huge amounts of computing power.*
  - *Architectures can be complex and often must be highly tailored to a specific application.*
  - *The resulting models may not be easily interpretable.*

Deep learning software for Python

- **Keras** https://keras.io/
- **Lasagne** https://lasagne.readthedocs.io/en/latest/

- **TensorFlow** https://www.tensorflow.org/
- **Theano** http://deeplearning.net/software/theano/

- **Libraries support high-performance computation via GPUs.**

Applied Machine Learning

Data Leakage

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Data Leakage

- **When the data you're using to train contains information about what you're trying to predict.**
- **Introducing information about the target during training that would not legitimately be available during actual use.**
- **Obvious examples:**
  - *Including the label to be predicted as a feature*
  - *Including test data with training data*
- **If your model performance is too good to be true, it probably is and likely due to "giveaway" features.**

More subtle examples of data leakage

- **Prediction target: will user stay on a site, or leave?**
  - *Giveaway feature: total session length, based on information about future page visits*
- **Predicting if a user on a financial site is likely to open an account**
  - *An account number field that's only filled in once the user does open an account.*
- **Diagnostic test to predict a medical condition**
  - *The existing patient dataset contains a binary variable that happens to mark whether they had surgery for that condition.*
  - *Combinations of missing diagnosis codes that are not be available while the patient's condition was still being studied.*
  - *The patient ID could contain information about specific diagnosis paths (e.g. for routine visit vs specialist).*
- **Any of these leaked features is highly predictive of the target, but not legitimately available at the time prediction needs to be done.**

Other examples of data leakage

Leakage in training data:

- Performing data preprocessing using parameters or results from analyzing the entire dataset: Normalizing and rescaling, detecting and removing outliers, estimating missing values, feature selection.
- Time-series datasets: using records from the future when computing features for the current prediction.
- Errors in data values/gathering or missing variable indicators (e.g. the special value 999) can encode information about missing data that reveals information about the future.

Leakage in features:

- Removing variables that are not legitimate without also removing variables that encode the same or related information (e.g. diagnosis info may still exist in patient ID).
- Reversing of intentional randomization or anonymization that reveals specific information about e.g. users not legitimately available in actual use.

Any of the above could be present in any external data joined to the training set.

Detecting data leakage

- **Before building the model**
  - *Exploratory data analysis to find surprises in the data*
  - *Are there features very highly correlated with the target value?*
- **After building the model**
  - *Look for surprising feature behavior in the fitted model.*
  - *Are there features with very high weights, or high information gain?*
  - *Simple rule-based models like decision trees can help with features like account numbers, patient IDs*
  - *Is overall model performance surprisingly good compared to known results on the same dataset, or for similar problems on similar datasets?*
- **Limited real-world deployment of the trained model**
  - *Potentially expensive in terms of development time, but more realistic*
  - *Is the trained model generalizing well to new data?*

Minimizing Data Leakage

- **Perform data preparation within each cross-validation fold separately**
  - *Scale/normalize data, perform feature selection, etc. within each fold separately, not using the entire dataset.*
  - *For any such parameters estimated on the training data, you must use those same parameters to prepare data on the corresponding held-out test fold.*
- **With time series data, use a timestamp cutoff**
  - *The cutoff value is set to the specific time point where prediction is to occur using current and past records.*
  - *Using a cutoff time will make sure you aren't accessing any data records that were gathered after the prediction time, i.e. in the future.*
- **Before any work with a new dataset, split off a final test validation dataset**
  - *… if you have enough data*
  - *Use this final test dataset as the very last step in your validation*
  - *Helps to check the true generalization performance of any trained models*