

# 3 - 2 - 1 - 0: Classifying Digits with R

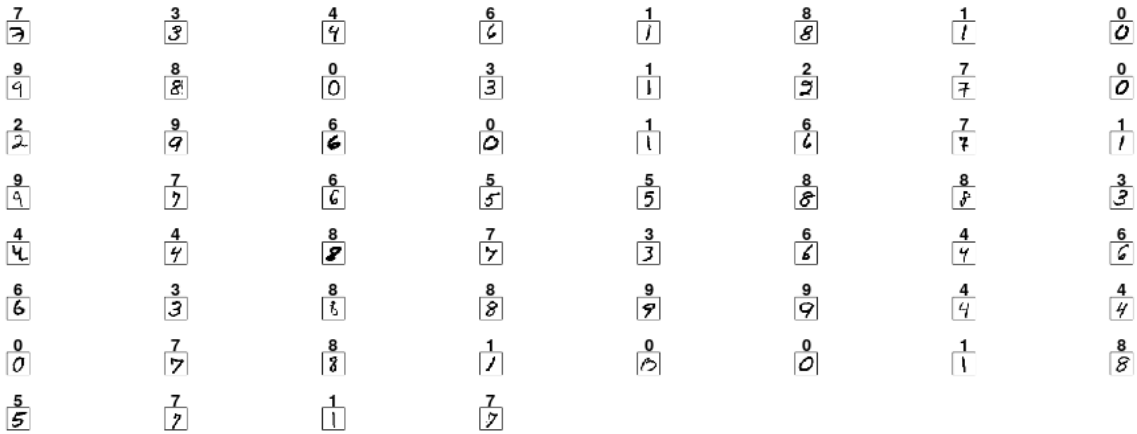
R for SQListas, a Continuation

# R for SQListas: Now that we're in the tidyverse ...

... what can we do now?

# Machine Learning

MNIST - the “Drosophila of Machine Learning” (attributed to Geoffrey Hinton)



# MNIST

- 60.000 train and 10.000 test examples of handwritten digits, 28x28 px
- download from [Yann LeCun's website](#)
- where you also find the “shootout of classifiers” ...

CLASSIFIER	PREPROCESSING	TEST ERROR RATE (%)	Reference
<b>Linear Classifiers</b>			
linear classifier (1-layer NN)	none	12.0	<a href="#">LeCun et al. 1998</a>
linear classifier (1-layer NN)	deskewing	8.4	<a href="#">LeCun et al. 1998</a>
pairwise linear classifier	deskewing	7.6	<a href="#">LeCun et al. 1998</a>
<b>K-Nearest Neighbors</b>			
K-nearest-neighbors, Euclidean (L2)	none	5.0	<a href="#">LeCun et al. 1998</a>
K-nearest-neighbors, Euclidean (L2)	none	3.09	<a href="#">Kenneth Wilder, U. Chicago</a>
K-nearest-neighbors, L3	none	2.83	<a href="#">Kenneth Wilder, U. Chicago</a>
K-nearest-neighbors, Euclidean (L2)	deskewing	2.4	<a href="#">LeCun et al. 1998</a>
K-nearest-neighbors, Euclidean (L2)	deskewing, noise removal, blurring	1.80	<a href="#">Kenneth Wilder, U. Chicago</a>
K-nearest-neighbors, L3	deskewing, noise removal, blurring	1.73	<a href="#">Kenneth Wilder, U. Chicago</a>
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 1 pixel shift	1.33	<a href="#">Kenneth Wilder, U. Chicago</a>
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 2 pixel shift	1.22	<a href="#">Kenneth Wilder, U. Chicago</a>

# The data

Use the R tensorflow library to load the data.  
Explanations, later ;-)

```
library(tensorflow)
datasets <- tf$contrib$learn$datasets
mnist <- datasets$mnist$read_data_sets("MNIST-data", one_hot = TRUE)

train_images <- mnist$train$images
train_labels <- mnist$train$labels

label_1 <- train_labels[1,]
image_1 <- train_images[1,]
```

# Images and labels

```
label_1
```

```
[1] 0 0 0 0 0 0 0 1 0 0
```

```
length(image_1)
```

```
[1] 784
```

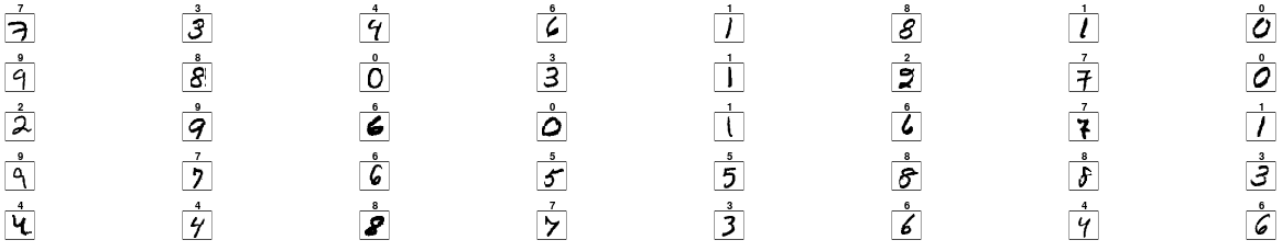
```
image_1[250:300]
```

```
[1] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.54901963  
[7] 0.98431379 0.99607849 0.99607849 0.99607849 0.99607849 0.99607849  
[13] 0.99607849 0.99607849 0.99607849 0.99607849 0.99607849 0.99607849  
[19] 0.99607849 0.99607849 0.99607849 0.74117649 0.09019608 0.00000000  
[25] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000  
[31] 0.00000000 0.00000000 0.00000000 0.88627458 0.99607849 0.81568635  
[37] 0.78039223 0.78039223 0.78039223 0.78039223 0.54509807 0.23921570  
[43] 0.23921570 0.23921570 0.23921570 0.23921570 0.50196081 0.87058830  
[49] 0.99607849 0.99607849 0.74117649
```

# Example images

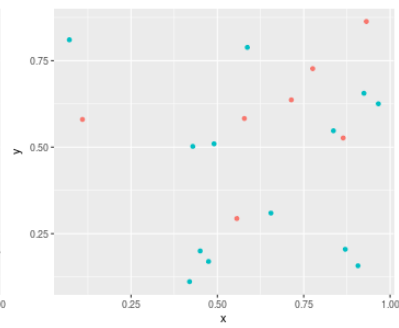
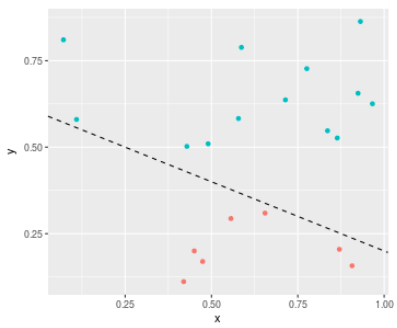
```
grayscale <- colorRampPalette(c('white','black'))
par(mar=c(1,1,1,1), mfrow=c(8,8),pty='s',xaxt='n',yaxt='n')

for(i in 1:40)
{
  z<-array(train_images[i,],dim=c(28,28))
  z<-z[,28:1] ##right side up
  image(1:28,1:28,z,main=which.max(train_labels[i,])-1,col=grayscale(256), , xlab="", ylab="")
}
```



# Classifying digits, try 1: Linear classifiers

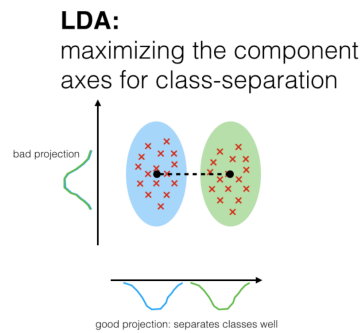
Are my data linearly separable?





# Maximizing between-class variance: Linear discriminant analysis (LDA)

LDA works by maximizing variance between classes.



Source: [http://sebastianraschka.com/Articles/2014\\_python\\_lda.html](http://sebastianraschka.com/Articles/2014_python_lda.html)

# Trying a linear classifier: Linear discriminant analysis (LDA)

```
# fit the model
lda_fit <- lda(X_train, y_train)

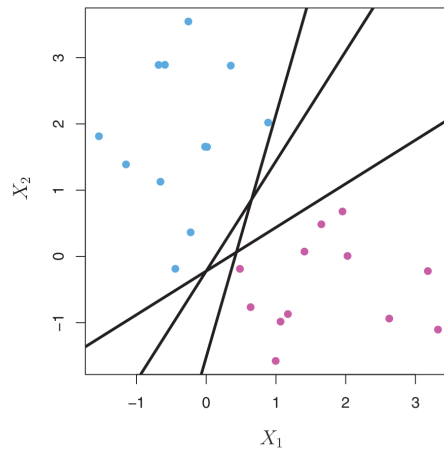
# model predictions for the test set
lda_pred <- predict(lda_fit, X_test)

# prediction accuracy
ct <- table(lda_pred$class, y_test)
sum(diag(prop.table(ct)))
```

```
[1] 0.8736
```

# Toward the (linear) SVM: Maximizing the margin (1)

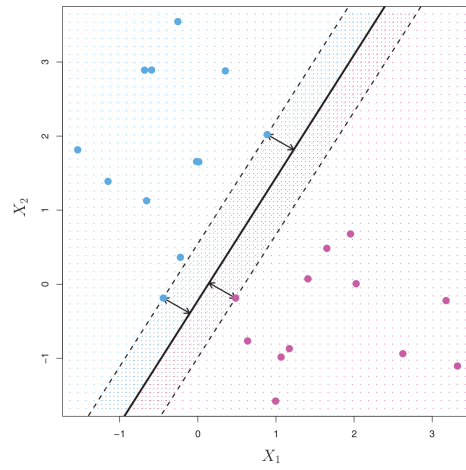
For linearly separable data, there are infinitely many ways to fit a separating line



Source: G. James, D. Witten, T. Hastie and R. Tibshirani, An Introduction to Statistical Learning, with applications in R

# Toward the (linear) SVM: Maximizing the margin (2)

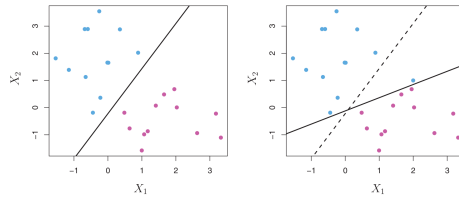
Redefine task: Maximal margin



Source: G. James, D. Witten, T. Hastie and R. Tibshirani, An Introduction to Statistical Learning, with applications in R

# Allowing for misclassification: Support Vector Classifier

Why allow for misclassifications?



Source: G. James, D. Witten, T. Hastie and R. Tibshirani, An Introduction to Statistical Learning, with applications in R

# Support Vector Classifier: Linear SVM

```
# fit the model
svm_fit_linear <- ksvm(x = X_train, y = y_train, type='C-svc', kernel='vanilladot', C=1, scale=FALSE)
```

Setting default kernel parameters

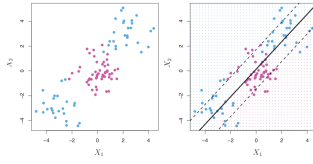
```
# model predictions for the test set
svm_pred <- predict(svm_fit_linear, X_test)

# prediction accuracy
ct <- table(svm_pred, y_test)
sum(diag(prop.table(ct)))
```

```
[1] 0.9393
```

# Going nonlinear: Support Vector Machine (1)

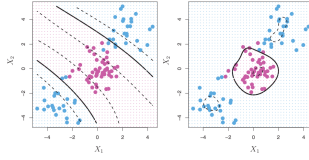
Why a linear classifier isn't enough



Source: G. James, D. Witten, T. Hastie and R. Tibshirani, An Introduction to Statistical Learning, with applications in R

# Going nonlinear: Support Vector Machine (2)

Non-linear kernels (polynomial resp. radial)



Source: G. James, D. Witten, T. Hastie and R. Tibshirani, An Introduction to Statistical Learning, with applications in R



# Support Vector Machine: RBF kernel

```
# fit the model
svm_fit_rbf <- ksvm(x = X_train, y = y_train, type='C-svc', kernel='rbf', C=1, scale=FALSE)

# model predictions for the test set
svm_pred <- predict(svm_fit_rbf, X_test)

# prediction accuracy
ct <- table(svm_pred, y_test)
sum(diag(prop.table(ct)))
```

```
[1] 0.9768
```

# Can this get any better?

Let's try neural networks!

# TensorFlow

AI library open sourced by Google

“If you can express your computation as a data flow graph, you can use TensorFlow.”

- implemented in C++, with C++ and Python APIs
- computations are graphs
- nodes are operations
- edges specify input to / output from operations - the *Tensors* (multidimensional matrices)
- the graph is just a spec - to make anything happen, execute it in a *Session*
- a *Session* places and runs a graph on a *Device* (GPU, CPU)

# TensorFlow in R

tensorflow R package: [Installation guide and tutorials](#)

Let's get started!

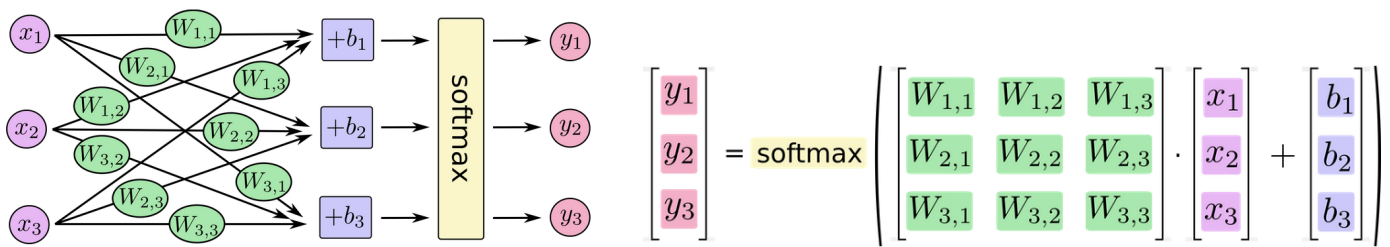
```
library(tensorflow)
```

# MNIST with TensorFlow: Load data and declare placeholders

```
datasets <- tf$contrib$learn$datasets
mnist <- datasets$mnist$read_data_sets("MNIST-data", one_hot = TRUE)

# images are 55000 * 784
x <- tf$placeholder(tf$float32, shape(NULL, 784L))
# labels are 55000 * 10
y_ <- tf$placeholder(tf$float32, shape(NULL, 10L))
```

# First, a shallow neural network



From: [TensorFlow tutorial](#)

- no hidden layers, just input layer and output layer
- softmax activation function

# Shallow network: Configuration

```
# weight matrix is 784 * 10
W <- tf$Variable(tf$zeros(shape(784L, 10L)))
# bias is 10 * 1
b <- tf$Variable(tf$zeros(shape(10L)))
# initialize variables

# y_hat
y <- tf$nn$softmax(tf$matmul(x,W) + b)
# loss function
cross_entropy <- tf$reduce_mean(-tf$reduce_sum(y_ * tf$log(y), reduction_indices=1L))
# specify optimization method and step size
optimizer <- tf$train$GradientDescentOptimizer(0.5)
train_step <- optimizer$minimize(cross_entropy)
```

# Shallow network: Training

```
sess = tf$InteractiveSession()
sess$run(tf$initialize_all_variables())

for (i in 1:1000) {
  batches <- mnist$train$next_batch(100L)
  batch_xs <- batches[[1]]
  batch_ys <- batches[[2]]
  sess$run(train_step, feed_dict = dict(x = batch_xs, y_ = batch_ys))
}
```



# Shallow Network: Evaluate

```
correct_prediction <- tf$equal(tf$argmax(y, 1L), tf$argmax(y_, 1L))
accuracy <- tf$reduce_mean(tf$cast(correct_prediction, tf$float32))

# actually evaluate training accuracy
sess$run(accuracy, feed_dict=dict(x = mnist$train$images, y_ = mnist$train$labels))
```

```
[1] 0.9167272
```

```
# and test accuracy
sess$run(accuracy, feed_dict=dict(x = mnist$test$images, y_ = mnist$test$labels))
```

```
[1] 0.918
```

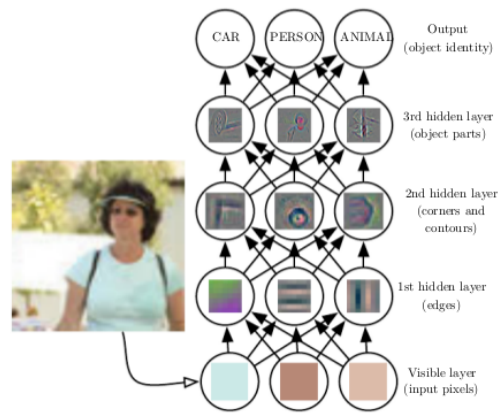
Hm. Accuracy's worse than with non-linear SVM...

Bit disappointing right?

Anything we can do?

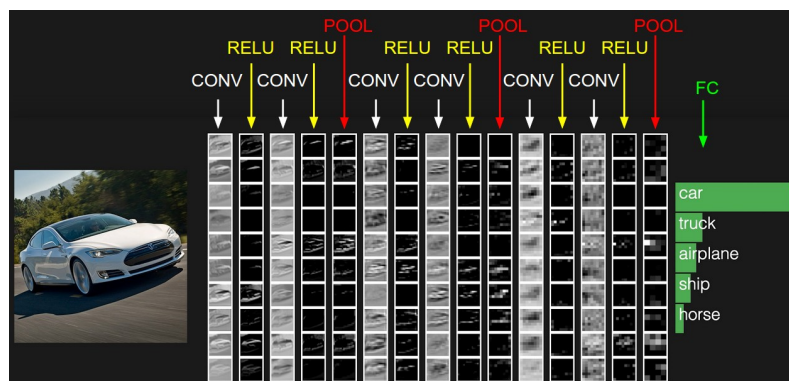
# Getting in deeper: Deep Learning

Discerning features, a layer at a time



Source: [Goodfellow et al. 2016, Deep Learning](#)

# Getting in deeper: Convnets (Convolutional Neural Networks)



Source: <http://cs231n.github.io/convolutional-networks/>

# Layer 1: Convolution and ReLU (1)

```
# template to initialize weights with a small amount of noise for symmetry breaking and to prevent 0 gradients
weight_variable <- function(shape) {
  initial <- tf$truncated_normal(shape, stddev=0.1)
  tf$Variable(initial)
}
# template to initialize bias to small positive value to avoid "dead neurons"
bias_variable <- function(shape) {
  initial <- tf$constant(0.1, shape=shape)
  tf$Variable(initial)
}

# compute 32 feature maps for each 5x5 patch
# we have just 1 channel
# so weights shape is: height, width, number of input channels, number of output channels
W_conv1 <- weight_variable(shape(5L, 5L, 1L, 32L))
# shape for bias: number of output channels
b_conv1 <- bias_variable(shape(32L))

# reshape x from 2d to 4d tensor with dimensions batch size, width, height, number of color channels
x_image <- tf$reshape(x, shape(-1L, 28L, 28L, 1L))
```

# Layer 1: convolution and ReLU (2)

```
# template to define convolutional layer
# tf$nn$conv2d parameters: input tensor, kernel tensor, strides, padding
# input tensor has shape [batch size, in_height, in_width, in_channels] (NHWC)
# kernel tensor has shape [filter_height, filter_width, in_channels, out_channels]
conv2d <- function(x, W) {
  tf$nn$conv2d(x, W, strides=c(1L, 1L, 1L, 1L), padding='SAME')
}
# perform convolution and ReLU activation
# output shape is batch size, 28, 28, 32
h_conv1 <- tf$nn$relu(conv2d(x_image, W_conv1) + b_conv1)
```

## Layer 2: Max pooling

```
# template to define max pooling over 2x2 regions
max_pool_2x2 <- function(x) {
  tf$nn$max_pool(
    x,
    ksize=c(1L, 2L, 2L, 1L),
    strides=c(1L, 2L, 2L, 1L),
    padding='SAME')
}

# output shape is batch size , 14, 14, 32
h_pool1 <- max_pool_2x2(h_conv1)
```

# Layer 3: convolution and ReLU

```
# next feature map is 5*5, takes 32 channels, produces 64 channels - size weights accordingly
W_conv2 <- weight_variable(shape = shape(5L, 5L, 32L, 64L))
b_conv2 <- bias_variable(shape = shape(64L))
# shape is ?, 14, 14, 64
h_conv2 <- tf$nn$relu(conv2d(h_pool1, W_conv2) + b_conv2)
```



## Layer 4: Max pooling

```
# output shape is batch size, 7, 7, 64  
h_pool2 <- max_pool_2x2(h_conv2)
```

# Layer 5: Densely connected layer

```
# bring together all feature maps
# weights shape: 3136, 1024 (fully connected)
W_fc1 <- weight_variable(shape(7L * 7L * 64L, 1024L))
b_fc1 <- bias_variable(shape(1024L))
# reshape input: batch size, 3136
h_pool2_flat <- tf$reshape(h_pool2, shape(-1L, 7L * 7L * 64L))

# matrix multiply and ReLU
# new shape: batch size, 1024
h_fc1 <- tf$nn$relu(tf$matmul(h_pool2_flat, W_fc1) + b_fc1)

#dropout
keep_prob <- tf$placeholder(tf$float32)
# shape: ?, 1024
h_fc1_drop <- tf$nn$dropout(h_fc1, keep_prob)
```

# Layer 5: Softmax

```
W_fc2 <- weight_variable(shape(1024L, 10L))
b_fc2 <- bias_variable(shape(10L))
# output shape: batch size, 10
y_conv <- tf$nn$softmax(tf$matmul(h_fc1_drop, W_fc2) + b_fc2)
```

# CNN: Define loss function and optimization algorithm

```
cross_entropy <- tf$reduce_mean(-tf$reduce_sum(y_ * tf$log(y_conv), reduction_indices=1L))
train_step <- tf$train$AdamOptimizer(1e-4)$minimize(cross_entropy)
correct_prediction <- tf$equal(tf$argmax(y_conv, 1L), tf$argmax(y_, 1L))
accuracy <- tf$reduce_mean(tf$cast(correct_prediction, tf$float32))
```

# CNN: Train network

```
for (i in 1:2000) {  
  batch <- mnist$train$next_batch(50L)  
  if (i %% 250 == 0) {  
    train_accuracy <- accuracy$eval(feed_dict = dict(  
      x = batch[[1]], y_ = batch[[2]], keep_prob = 1.0))  
    cat(sprintf("step %d, training accuracy %g\n", i, train_accuracy))  
  }  
  train_step$run(feed_dict = dict(  
    x = batch[[1]], y_ = batch[[2]], keep_prob = 0.5), session=sess)  
}
```

```
step 250, training accuracy 0.88  
step 500, training accuracy 0.9  
step 750, training accuracy 0.9  
step 1000, training accuracy 0.96  
step 1250, training accuracy 0.98  
step 1500, training accuracy 1  
step 1750, training accuracy 1  
step 2000, training accuracy 0.98
```

# CNN: Accuracy on test set

```
test_accuracy <- accuracy$eval(feed_dict = dict(  
  x = mnist$test$images, y_ = mnist$test$labels, keep_prob = 1.0))  
cat(sprintf("test accuracy %g", train_accuracy))
```

```
test accuracy 0.98
```

Now could play around with the configuration to get even higher accuracy ...

... but that will have to be another time ...

Thanks for your attention!!