R Programming for Statistics and Machine Learning

Nikolay Oskolkov, MRG Group Leader, LIOS, Riga, Latvia

R course, 05.02.2026

@NikolayOskolkov

@oskolkov.bsky.social

Personal homepage:
https://nikolay-oskolkov.com

Image generated by ChatGPT

Brief introduction: who am I

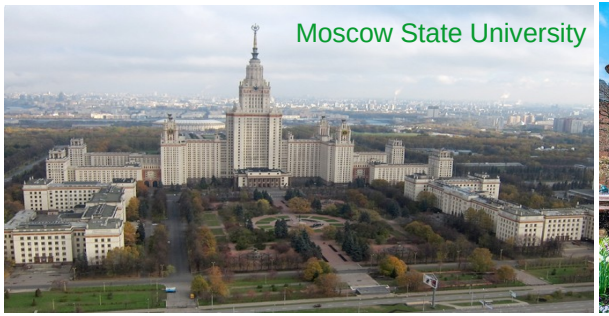
2007 PhD in theoretical physics in Moscow, Russia

2011 medical genomics at Lund University, Sweden

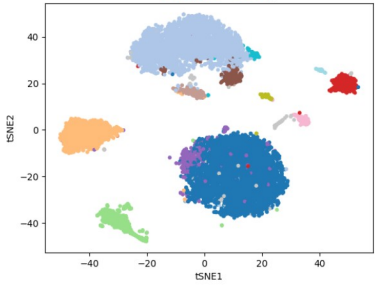
2016 bioinformatician at NBIS SciLifeLab, Sweden

2025 Metabolic Research Group leader, LIOS, Latvia

50+ publications; h-index = 25; 4,000+ citations



Single cell



Biomedical data integration

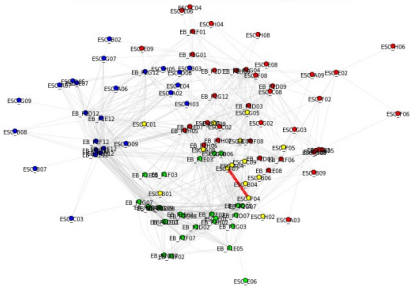
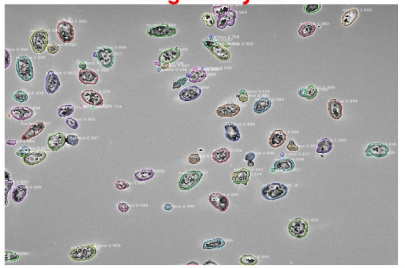
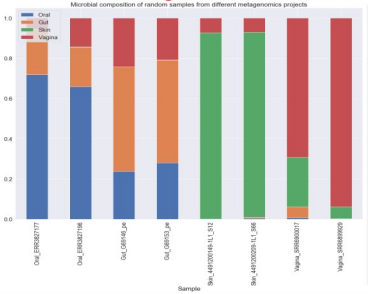


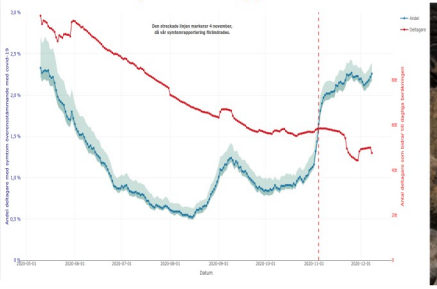
Image analysis



Metagenomics



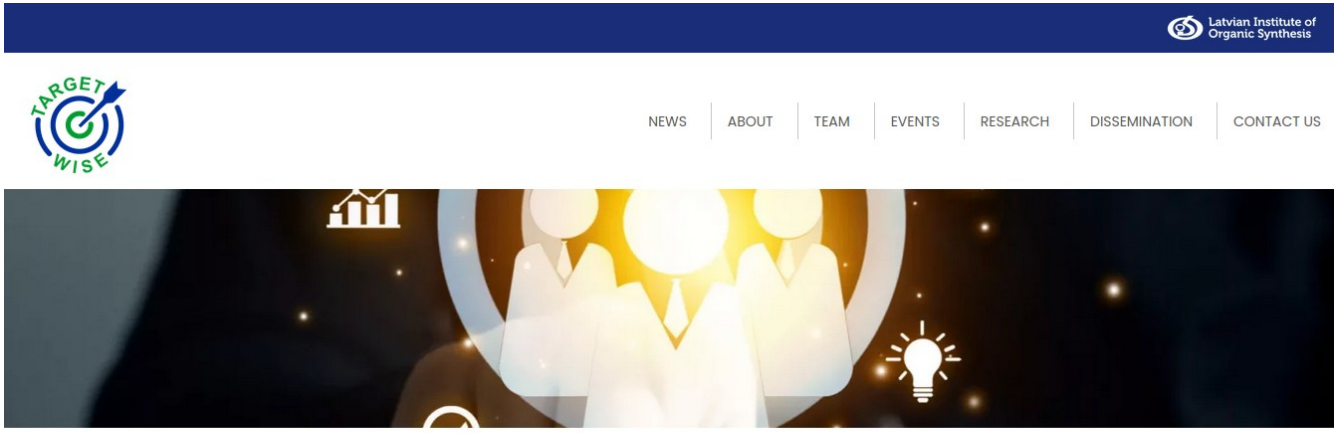
Epidemiology



Ancient microbiome

Metabolic Research Group at LIOS

Latvian Institute of Organic Synthesis



PhD. Nikolay Oskolkov
Group Leader (PI) of the Metabolic Research Group

1 more postdoctoral fellow and 1 PhD student to be hired

If you know anyone who might be interested, please contact me!

Kristina Grausa, PhD sudent in Metabolic Research Group

Daniel Rivas, MD, PhD in AI, postdoctoral fellow in Metabolic Research Group

About you

- Name
- University / Institute / Company
- Research interest(s)
- Previous experience with computational analysis and bioinformatics
- Motivation to join the course
- Expectations from the course

Housekeeping rules

- Please ask your questions anytime, it is ok to interrupt me, or you can raise your hand and ask. You can also send me your questions by email.

- The course includes 7 lectures (~1h each) and 8 practicals (~1h each), there are 30 min breaks after each lecture and each practical.

- During practicals, we will use Rmarkdown notebooks. At the end of each practical, I will use rendered html-versions of the notebooks to go through command lines with my explanations.

- The material is based on data and problems from computational biology, however the concepts discussed are general and can be applied for other types of data.

Introduction to Statistical Analysis in R

Nikolay Oskolkov, MRG Group Leader, LIOS, Riga, Latvia

R course, 05.02.2026

Image generated by ChatGPT

@NikolayOskolkov

@oskolkov.bsky.social

Personal homepage:
https://nikolay-oskolkov.com

Session content

Topics we'll cover in this session:

1) High-dimensional biological data and curse of dimensionality

2) Data-driven choice of statistical analysis

3) Basics of Frequentist statistics: pros and cons of p-value

4) Basics of Bayesian statistics: likelihood, prior and posterior

Biological data are high dimensional

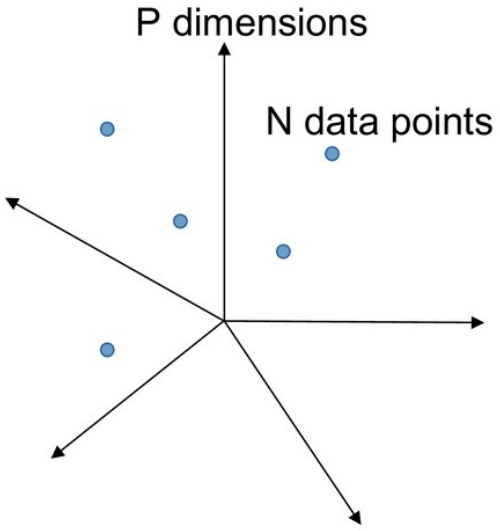
Statistical observations:
e.g. samples, cells etc.

N

P

$$\begin{pmatrix} 0 & 3 & 1 & 0 & 2 & 3 & 8 & 1 & 1 & 3 \\ 1 & 1 & 0 & 0 & 7 & 1 & 2 & 2 & 3 & 3 \\ 1 & 2 & 2 & 0 & 0 & 6 & 7 & 1 & 2 & 2 \\ 1 & 2 & 3 & 10 & 0 & 4 & 6 & 1 & 0 & 5 \\ 3 & 2 & 2 & 1 & 4 & 3 & 2 & 1 & 6 & 0 \\ 7 & 4 & 4 & 5 & 3 & 9 & 6 & 1 & 6 & 1 \\ 7 & 1 & 1 & 5 & 2 & 8 & 9 & 1 & 3 & 6 \\ 5 & 0 & 1 & 6 & 2 & 0 & 0 & 0 & 1 & 5 \\ 1 & 6 & 3 & 3 & 4 & 6 & 2 & 0 & 1 & 1 \\ 1 & 2 & 2 & 4 & 1 & 1 & 3 & 0 & 8 & 2 \end{pmatrix}$$

Features: genes, proteins, microbes, metabolites etc.



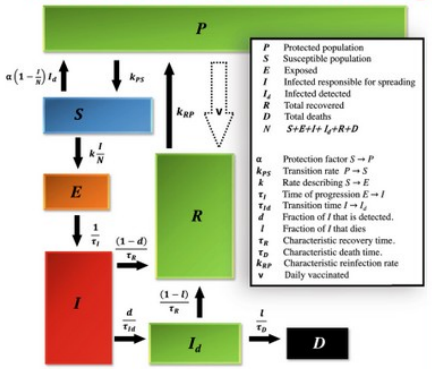
High Dimensional Data:
P >> N

For a robust statistical analysis, one should properly “sample” the P-dimensional space, hence large sample size is required, N >> P

Types of data analysis

Biology / Biomedicine

Mathematical modeling



N ⟶ 0

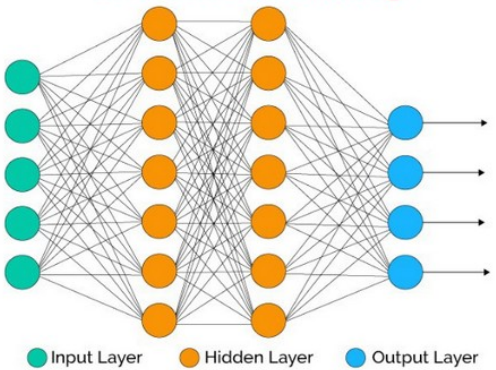
Bayesianism

N << P

Frequentism

N ≈ P

Machine Learning



N >> P

Hypothesis-driven ⟶ **Data-driven**

Amount of Data

The Curse of Dimensionality

Ex.1

$$Y = \alpha + \beta X$$

$$\beta = (X^T X)^{-1} X^T Y$$

$$(X^T X)^{-1} \sim \frac{1}{\det(X^T X)} \cdots \to \infty, \quad n << p$$

Ex.2 $$E[\hat{\sigma}^2] = \frac{n-p}{n}\sigma^2$$

Biased Maximum Likelihood variance estimator at n << p

Some peculiarities of Frequentist statistics

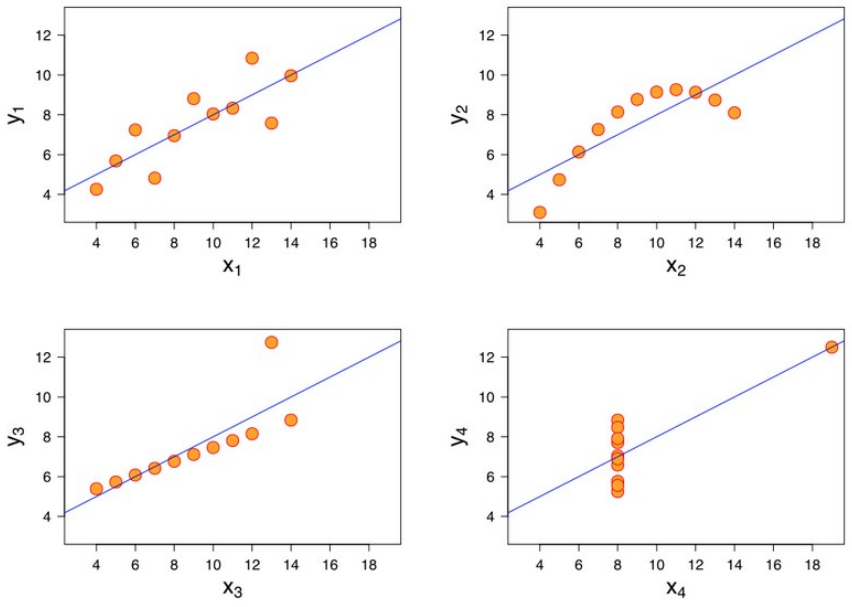
- based on Maximum Likelihood principle
- focus ~~too much~~ on summary statistics

$$L\left(x_i \mid \mu, \sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{\sum_{i=1}^{N}(x_i - \mu)^2}{2\sigma^2}}$$

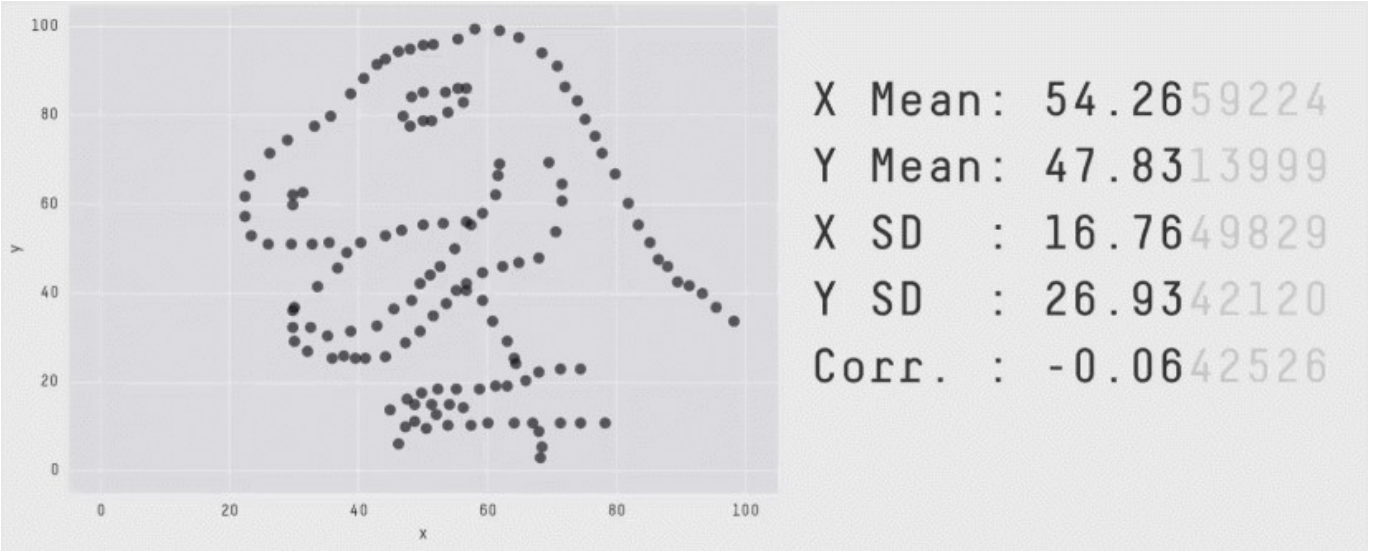
$$\frac{\partial L\left(x_i \mid \mu, \sigma^2\right)}{\partial \mu} = 0; \quad \frac{\partial L\left(x_i \mid \mu, \sigma^2\right)}{\partial \sigma^2} = 0$$

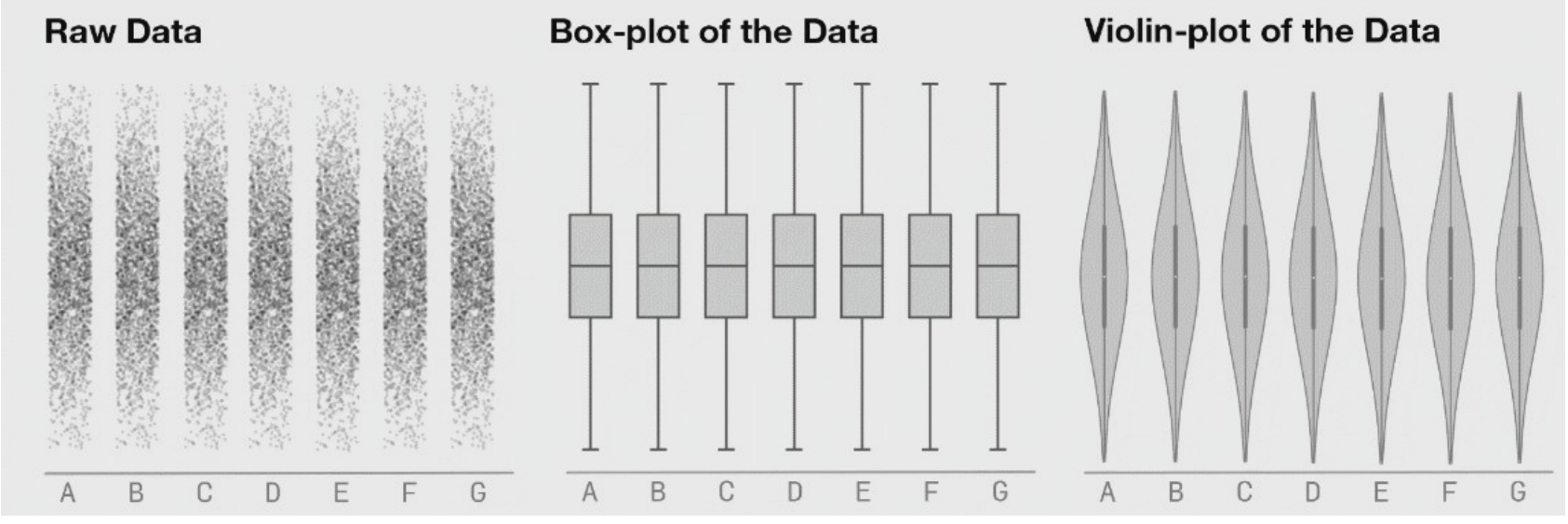
$$\mu = \frac{1}{N}\sum_{i=0}^{N} x_i \; - \text{mean estimator}$$

$$\sigma^2 = \frac{1}{N}\sum_{i=0}^{N}(x_i - \mu)^2 \; - \text{variance estimator}$$



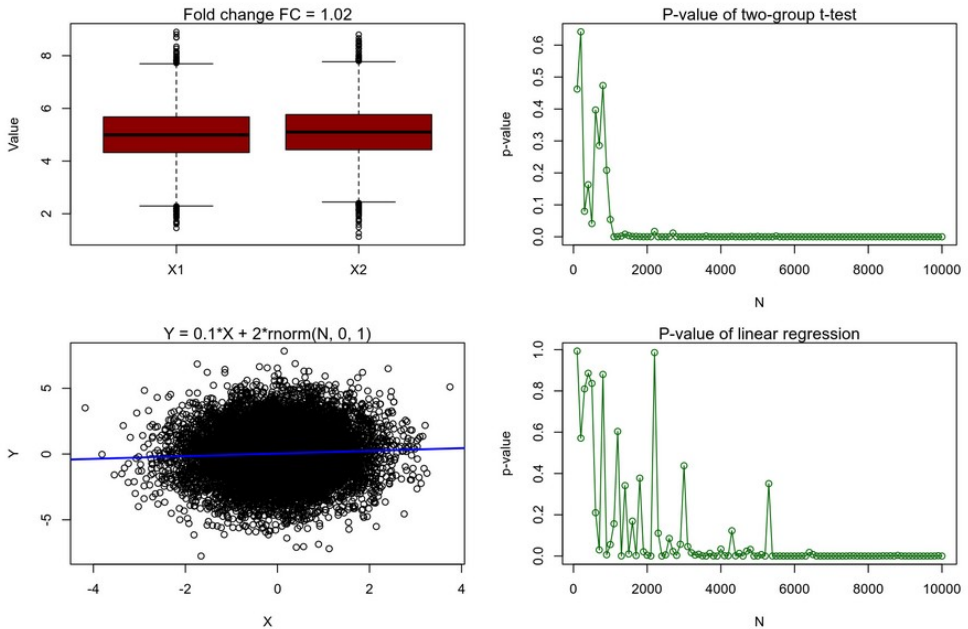
Summary statistics do not always reasonbly describe data (example: Anscombes quartet)


100
80
60
40
20
0
y
0
20
40
60
80
100
x
X Mean: 54.2659224
Y Mean: 47.8313999
X SD : 16.7649829
Y SD : 26.9342120
Corr. : -0.0642526


Raw Data
Box-plot of the Data
Violin-plot of the Data
A B C D E F G
A B C D E F G
A B C D E F G

Frequentist statistics: focus ~~to much~~ on p-values

```r
FC<-1.02; x_mean<-5; x_sd<-1; N_vector<-seq(from=100,to=10000,by=100); pvalue_t<-vector(); pvalue_lm<-vector()
for(N in N_vector)
{
  x1 <- rnorm(N, x_mean, x_sd); x2 <- rnorm(N, x_mean*FC, x_sd)
  t_test_res<-t.test(x1, x2); pvalue_t <- append(pvalue_t, t_test_res$p.value)

  x <- rnorm(N, 0, 1); y <- 0.1*x+2*rnorm(N, 0, 1)
  lm_res <- summary(lm(y~x)); pvalue_lm <- append(pvalue_lm, lm_res$coefficients[2,4])
}
par(mfrow=c(2,2)); par(mar = c(5, 5, 1, 1))
boxplot(x1, x2, names=c("X1","X2"), ylab="Value", col="darkred"); mtext("Fold change FC = 1.02")
plot(pvalue_t~N_vector,type='o',xlab="N",ylab="p-value",col="darkgreen"); mtext("P-value of two-group t-test")
plot(y~x, xlab="X", ylab="Y"); abline(lm(y~x), col="blue", lwd=2); mtext("Y = 0.1*X + 2*rnorm(N, 0, 1)")
plot(pvalue_lm~N_vector,type='o',xlab="N",ylab="p-value",col="darkgreen"); mtext("P-value of linear regression")
```



Questionable whether p-value is a best metric for ranking features (biomarkers)

Frequentist statistics struggles with high-dimensional data

```r
n <- 20 # number of samples
p <- 2  # number of features / dimensions
Y <- rnorm(n)
X <- matrix(rnorm(n * p), n, p)
summary(lm(Y ~ X))
```

```
Call:
lm(formula = Y ~ X)

Residuals:
    Min      1Q  Median      3Q     Max
-2.0522 -0.6380  0.1451  0.3911  1.8829

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.14950    0.22949   0.651    0.523
X1          -0.09405    0.28245  -0.333    0.743
X2          -0.11919    0.24486  -0.487    0.633

Residual standard error: 1.017 on 17 degrees of freedom
Multiple R-squared:  0.02204,   Adjusted R-squared:  -0.09301
F-statistic: 0.1916 on 2 and 17 DF,  p-value: 0.8274
```

Going to higher dimensions →

```r
n <- 20 # number of samples
p <- 10 # number of features / dimensions
Y <- rnorm(n)
X <- matrix(rnorm(n * p), n, p)
summary(lm(Y ~ X))
```

```
Call:
lm(formula = Y ~ X)

Residuals:
    Min      1Q  Median      3Q     Max
-1.0255 -0.4320  0.1056  0.4493  1.0617

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.54916    0.26472   2.075   0.0679 .
X1           0.30013    0.21690   1.384   0.1998
X2           0.68053    0.27693   2.457   0.0363 *
X3          -0.10675    0.26010  -0.410   0.6911
X4          -0.21367    0.33690  -0.634   0.5417
X5          -0.19123    0.31881  -0.600   0.5634
X6           0.81074    0.25221   3.214   0.0106 *
X7           0.09634    0.24143   0.399   0.6992
X8          -0.29864    0.19004  -1.571   0.1505
X9          -0.78175    0.35408  -2.208   0.0546 .
X10          0.83736    0.36936   2.267   0.0496 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8692 on 9 degrees of freedom
Multiple R-squared:  0.6592,    Adjusted R-squared:  0.2805
F-statistic: 1.741 on 10 and 9 DF,  p-value: 0.2089
```

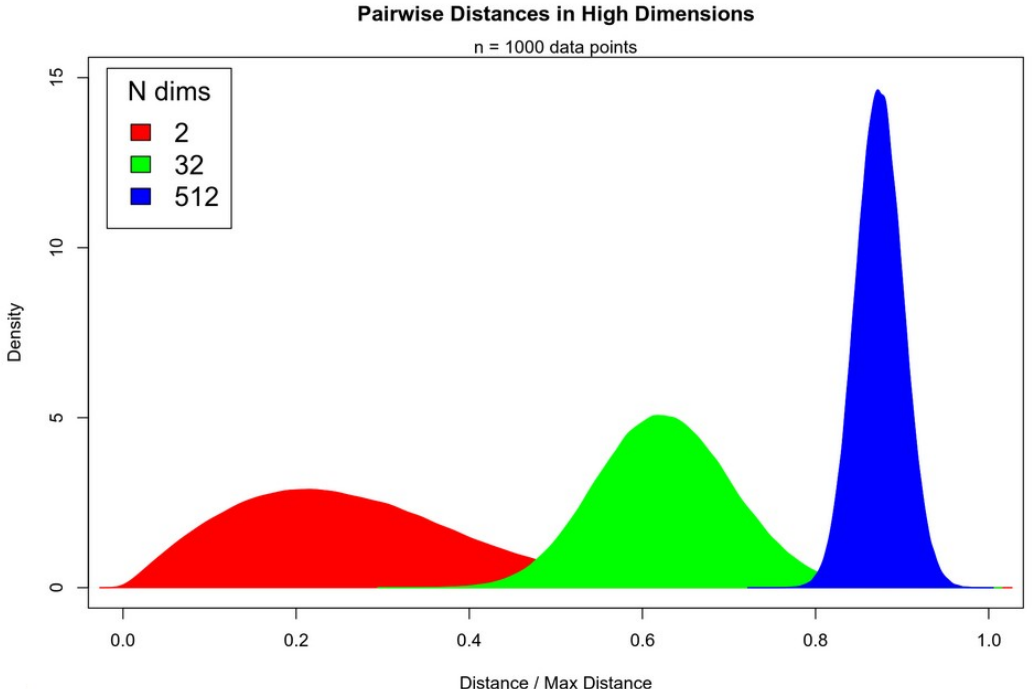
Going to even higher dimensions →

```r
n <- 20 # number of samples
p <- 20 # number of features / dimensions
Y <- rnorm(n)
X <- matrix(rnorm(n * p), n, p)
summary(lm(Y ~ X))
```

```
Call:
lm(formula = Y ~ X)

Residuals:
ALL 20 residuals are 0: no residual degrees of freedom!

Coefficients: (1 not defined because of singularities)
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.34889        NaN     NaN      NaN
X1           0.66218        NaN     NaN      NaN
X2           0.76212        NaN     NaN      NaN
X3          -1.35033        NaN     NaN      NaN
X4          -0.57487        NaN     NaN      NaN
X5           0.02142        NaN     NaN      NaN
X6           0.40290        NaN     NaN      NaN
X7           0.03313        NaN     NaN      NaN
X8          -0.31983        NaN     NaN      NaN
X9          -0.92833        NaN     NaN      NaN
X10          0.18091        NaN     NaN      NaN
X11         -1.37618        NaN     NaN      NaN
X12          2.11438        NaN     NaN      NaN
X13         -1.75103        NaN     NaN      NaN
X14         -1.55073        NaN     NaN      NaN
X15          0.01112        NaN     NaN      NaN
X16         -0.50943        NaN     NaN      NaN
X17         -0.47576        NaN     NaN      NaN
X18          0.31793        NaN     NaN      NaN
X19          1.43615        NaN     NaN      NaN
X20               NA         NA      NA       NA

Residual standard error: NaN on 0 degrees of freedom
Multiple R-squared:      1, Adjusted R-squared:    NaN
F-statistic:   NaN on 19 and 0 DF,  p-value: NA
```

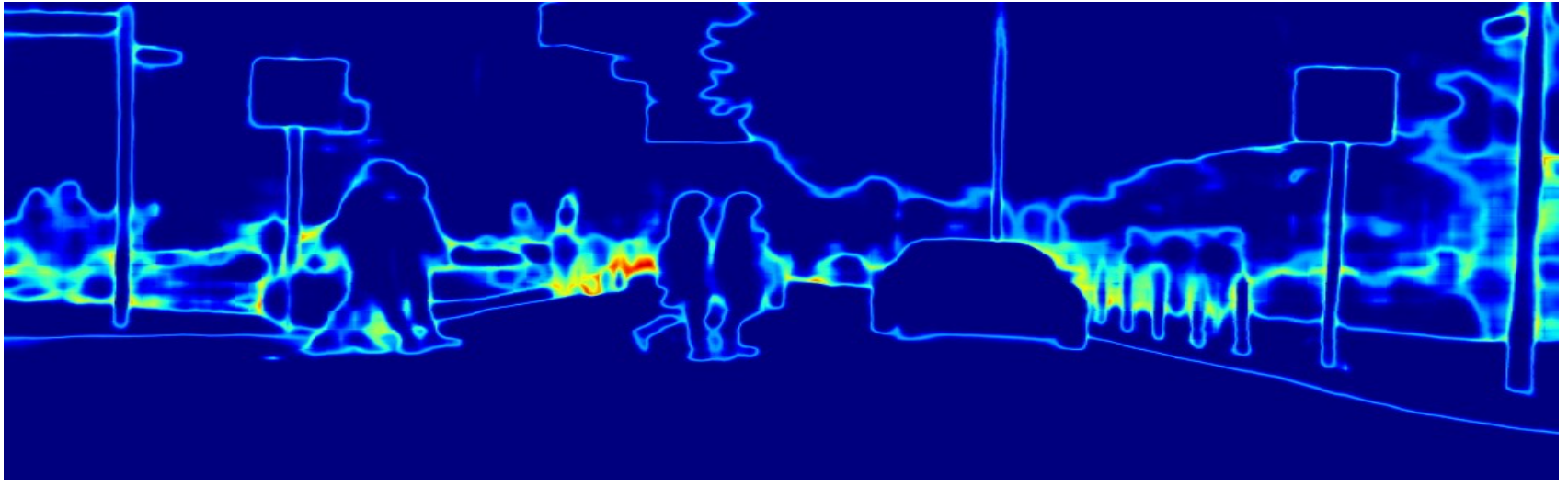
Equidistant points in high dimensions

```r
n <- 1000; p <- c(2, 32, 512); pair_dist <- list()
for(i in 1:length(p)) {
  X <- matrix(rnorm(n * p[i]), n, p[i])
  pair_dist[[i]] <- as.vector(dist(X));
  pair_dist[[i]] <- pair_dist[[i]] / max(pair_dist[[i]])
}
```

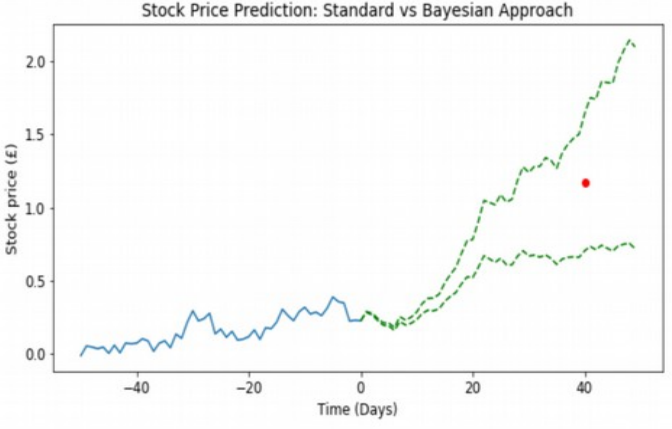


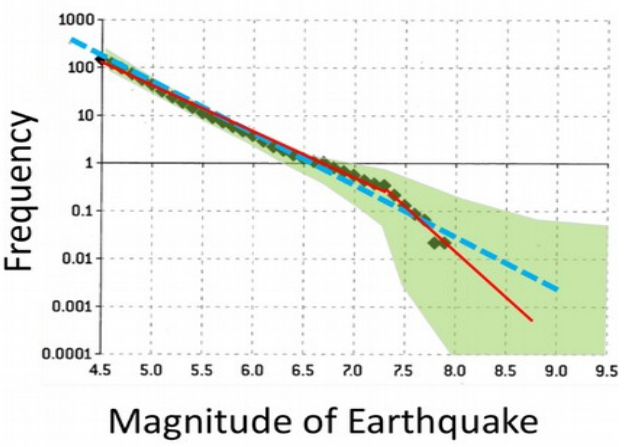
- Data points in high dimensions:
  - move **away** from each other
  - become **equidistant** and similar
- Impossible to see differences between cases and controls

When Frequentist stats are not good enough



Intelligence is to know how much you do not know





Regularizations are priors in Bayesian statistics

$$Y = \beta_1 X_1 + \beta_2 X_2 + \epsilon; \quad Y \sim N(\beta_1 X_1 + \beta_2 X_2, \sigma^2) \equiv \mathrm{L}(\mathrm{Y} \mid \beta_1, \beta_2)$$

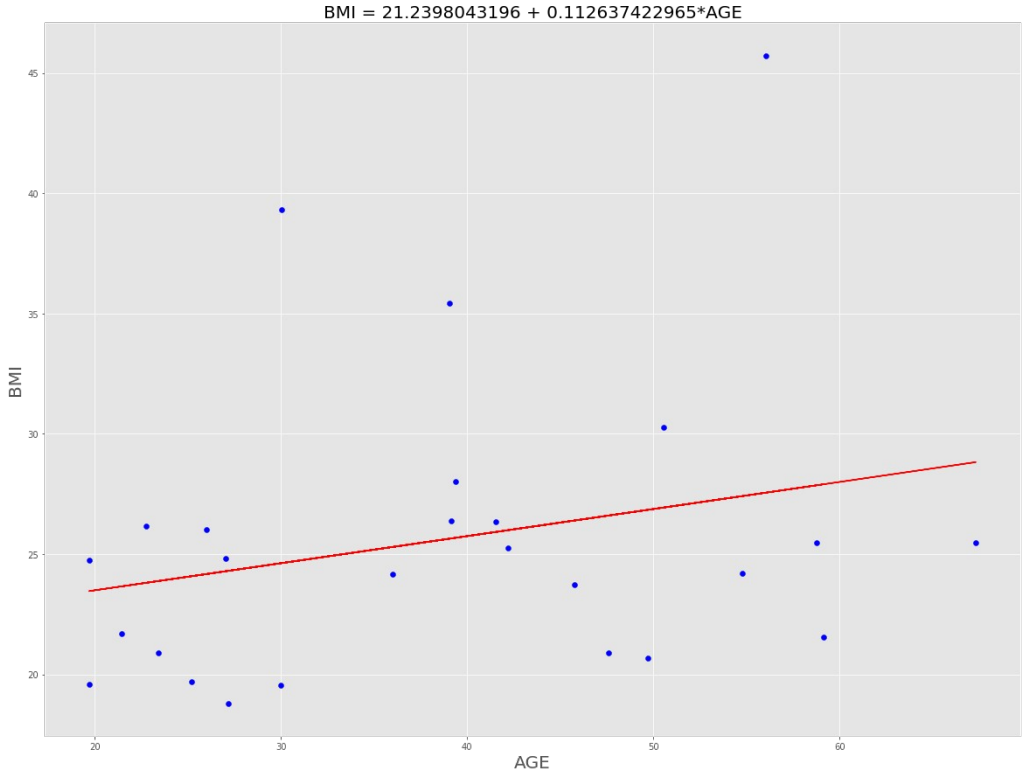
- **Maximum Likelihood** principle: maximize probability to observe data given parameters:

$$\mathrm{L}(\mathrm{Y} \mid \beta_1, \beta_2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(\mathrm{Y} - \beta_1 \mathrm{X}_1 - \beta_2 \mathrm{X}_2)^2}{2\sigma^2}}$$

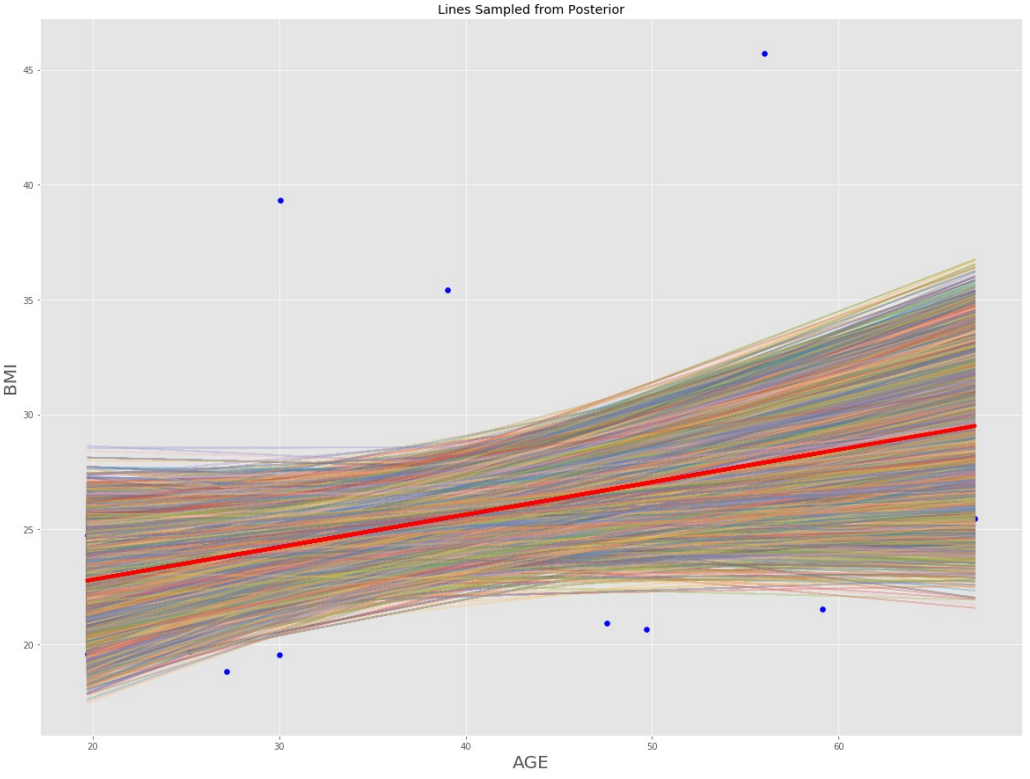
- **Bayes theorem**: maximize posterior probability of observing parameters given data:

$$\mathrm{Posterior}(\mathrm{params} \mid \mathrm{data}) = \frac{\mathrm{L}(\mathrm{data} \mid \mathrm{params}) * \mathrm{Prior}(\mathrm{params})}{\int \mathrm{L}(\mathrm{data} \mid \mathrm{params}) * \mathrm{Prior}(\mathrm{params})\, \mathrm{d}(\mathrm{params})}$$

Frequentist vs. Bayesian linear fitting

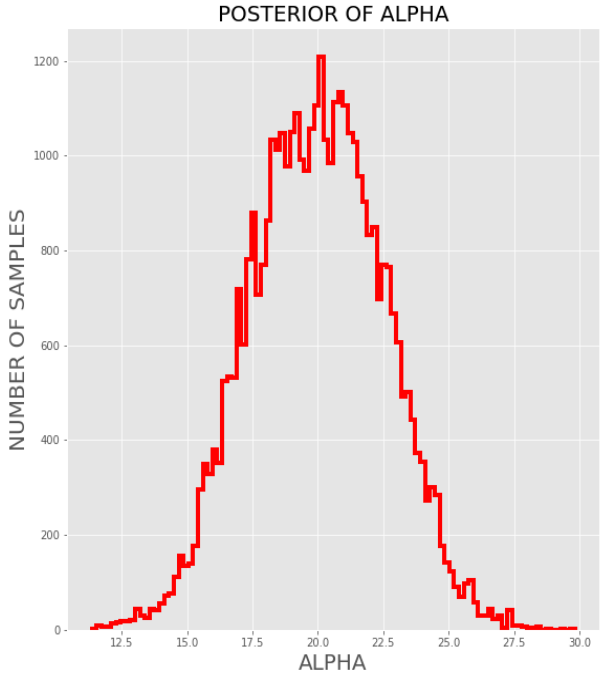


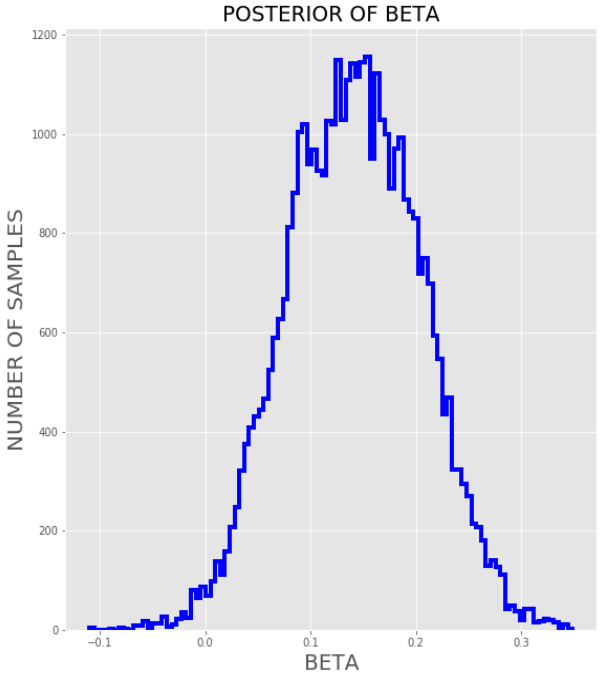
Fitting Frequentist linear model

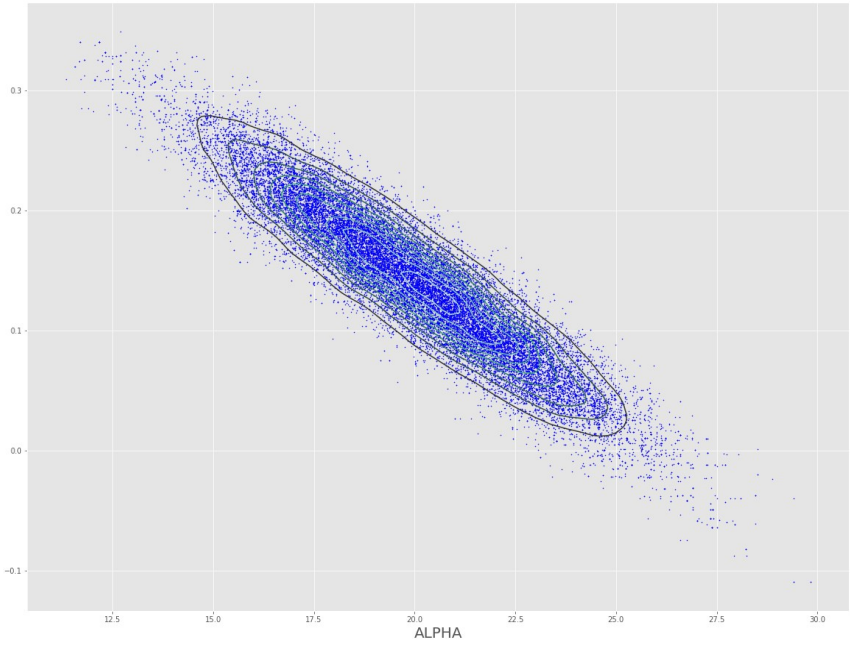


Fitting Bayesian linear model

Drawing possible fits from Posterior







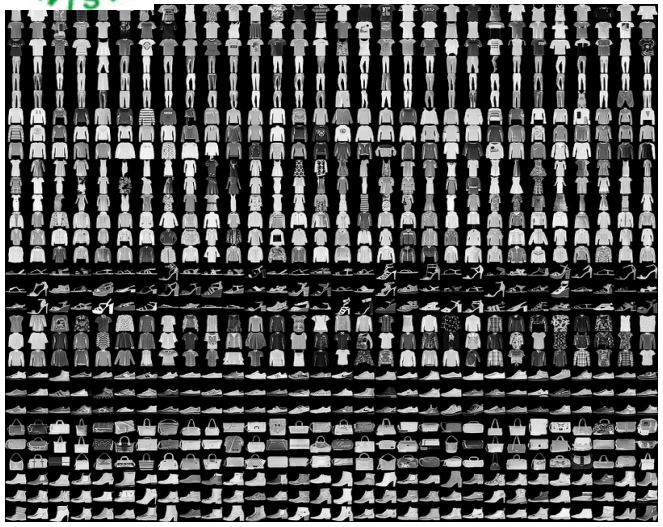
Slope and intersect are not fixed values any more

MCMC can draw infinite number of fits from proximity of optimal slope and intercept parameters

Frequentist image recognition

Latvian Institute of Organic Synthesis

Fashion MNIST



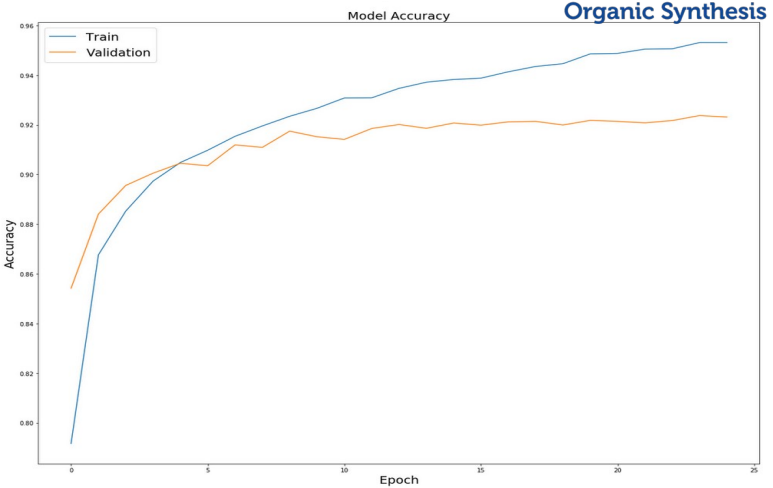
```
In [24]: # normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0

In [25]: # one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
print(num_classes)
10

In [27]: # Create the model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(1, 28, 28), padding='same', activation='relu',
                 kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), padding='same', activation='relu',
                 kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

# Compile model
epochs = 25
lrate = 0.01
decay = lrate/epochs
sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())
```

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 32, 28, 28)	320
dropout_7 (Dropout)	(None, 32, 28, 28)	0
conv2d_9 (Conv2D)	(None, 32, 28, 28)	9248
max_pooling2d_4 (MaxPooling2	(None, 32, 14, 14)	0
flatten_4 (Flatten)	(None, 6272)	0
dense_7 (Dense)	(None, 512)	3211776
dropout_8 (Dropout)	(None, 512)	0
dense_8 (Dense)	(None, 10)	5130

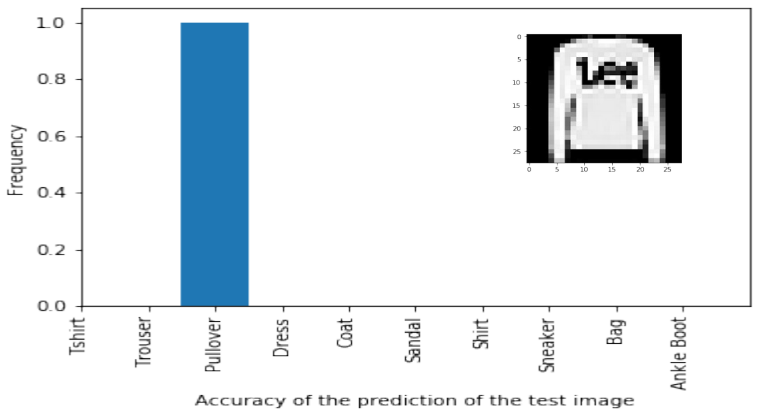
Total params: 3,226,474
Trainable params: 3,226,474
Non-trainable params: 0

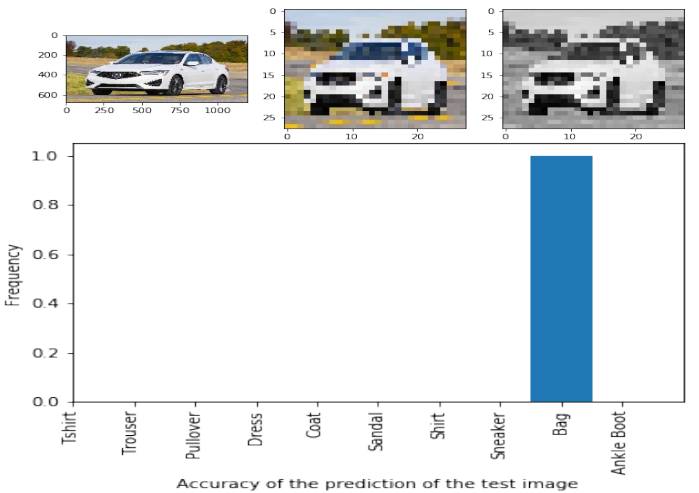
None

```
In [28]: # Fit the model
#model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)
history = model.fit(X_train, y_train, epochs = epochs, verbose = 1, validation_split = 0.25,
                    batch_size = 32, shuffle = True)
Train on 45000 samples, validate on 15000 samples
Epoch 1/25
45000/45000 [==============================] - 1158s 26ms/step - loss: 0.5762 - acc: 0.7917 - val_loss: 0.39
73 - val_acc: 0.8542
Epoch 2/25
45000/45000 [==============================] - 1124s 25ms/step - loss: 0.3643 - acc: 0.8676 - val_loss: 0.31
27 - val_acc: 0.8841
Epoch 3/25
45000/45000 [==============================] - 1158s 26ms/step - loss: 0.3129 - acc: 0.8853 - val_loss: 0.28
25 - val_acc: 0.8956
Epoch 4/25
45000/45000 [==============================] - 1609s 36ms/step - loss: 0.2813 - acc: 0.8973 - val_loss: 0.27
27 - val_acc: 0.9005
Epoch 5/25
45000/45000 [==============================] - 902s 20ms/step - loss: 0.2618 - acc: 0.9048 - val_loss: 0.258
8 - val_acc: 0.9045
Epoch 6/25
45000/45000 [==============================] - 936s 21ms/step - loss: 0.2451 - acc: 0.9098 - val_loss: 0.256
4 - val_acc: 0.9035
Epoch 7/25
```



Prediction



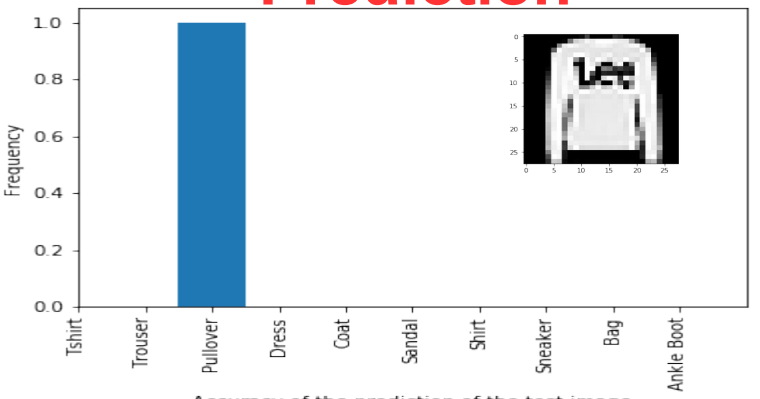


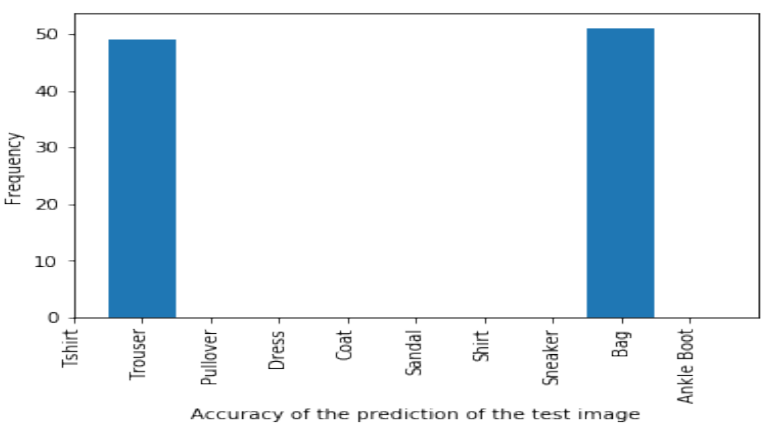
Bayesian image recognition

PyMC3, Edward, TensorFlow Probability

```
In [8]:
x_train = x_train.reshape((x_train.shape[0],D))
x_test = x_test.reshape((x_test.shape[0],D))
print(x_train.shape)
print(x_test.shape)

(60000, 784)
(10000, 784)

In [9]:
from keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
print(y_train.shape)
print(y_test.shape)

(60000, 10)
(10000, 10)

In [10]:
ed.set_seed(314159)
N = 100   # number of images in a minibatch.
D = D     # number of features.
K = 10    # number of classes.

# Create a placeholder to hold the data (in minibatches) in a TensorFlow graph.
x = tf.placeholder(tf.float32, [None, D])
# Normal(0,1) priors for the variables. Note that the syntax assumes TensorFlow 1.1.
w = Normal(loc=tf.zeros([D, K]), scale=tf.ones([D, K]))
b = Normal(loc=tf.zeros(K), scale=tf.ones(K))
# Categorical likelihood for classication.
y = Categorical(tf.matmul(x, w) + b)

In [11]:
# Contruct the q(w) and q(b). in this case we assume Normal distributions.
qw = Normal(loc=tf.Variable(tf.random_normal([D, K])),
            scale=tf.nn.softplus(tf.Variable(tf.random_normal([D, K]))))
qb = Normal(loc=tf.Variable(tf.random_normal([K])),
            scale=tf.nn.softplus(tf.Variable(tf.random_normal([K]))))

In [12]:
def generator(arrays, batch_size = N):
    starts = [0] * len(arrays)  # pointers to where we are in iteration
    while True:
        batches = []
        for i, array in enumerate(arrays):
            start = starts[i]
            stop = start + batch_size
            diff = stop - array.shape[0]
            if diff <= 0:
                batch = array[start:stop]
                starts[i] += batch_size
            else:
                batch = np.concatenate((array[start:], array[:diff]))
                starts[i] = diff
            batches.append(batch)
        yield batches
cifar10 = generator([x_train, y_train], N)

In [13]:
# We use a placeholder for the labels in anticipation of the traning data.
y_ph = tf.placeholder(tf.int32, [N])
# Define the VI inference technique, ie. minimise the KL divergence between q and p.
inference = ed.KLqp({w: qw, b: qb}, data={y: y_ph})
# Initialse the infernce variables
inference.initialize(n_iter=50000, n_print=100, scale={y: float(x_train.shape[0]) / N})
# We will use an interactive session.
sess = tf.InteractiveSession()
# Initialise all the vairables in the session.
tf.global_variables_initializer().run()
# Let the training begin. We load the data in minibatches and update the VI infernce using each new batch.
for _ in range(inference.n_iter):
    X_batch, Y_batch = next(cifar10)
    #X_batch = X_batch.reshape(N,-1)
    # TensorFlow method gives the label data in a one hot vetor format. We convert that into a single label.
    Y_batch = np.argmax(Y_batch, axis=1)
    info_dict = inference.update(feed_dict={x: X_batch, y_ph: Y_batch})
    inference.print_progress(info_dict)

50000/50000 [100%] ██████████████████████████████ Elapsed: 221s | Loss: 85453.266

In [14]:
# Generate samples the posterior and store them.
n_samples = 100
prob_lst = []
samples = []
w_samples = []
b_samples = []
for _ in range(n_samples):
    w_samp = qw.sample()
    b_samp = qb.sample()
    w_samples.append(w_samp)
    b_samples.append(b_samp)
    # Also compute the probabiliy of each class for each (w,b) sample.
    prob = tf.nn.softmax(tf.matmul(x_test, w_samp) + b_samp)
    prob_lst.append(prob.eval())
    sample = tf.concat([tf.reshape(w_samp,[-1]),b_samp],0)
    samples.append(sample.eval())

In [15]:
# Compute the accuracy of the model.
# For each sample we compute the predicted class and compare with the test labels.
# Predicted class is defined as the one which as maximum probability.
# We perform this test for each (w,b) in the posterior giving us a set of accuracies
# Finally we make a histogram of accuracies for the test data.
accy_test = []
for prob in prob_lst:
    y_trn_prd = np.argmax(prob, axis=1).astype(np.float32)
    acc = (y_trn_prd == np.argmax(y_test, axis=1)).mean()*100
    accy_test.append(acc)

plt.hist(accy_test)
plt.title("Histogram of prediction accuracies in the CIFAR10 test data")
plt.xlabel("Accuracy")
plt.ylabel("Frequency")
```

Prediction





Session summary

Take home messages of the session:

1) Biological data are high dimensional and this should be taken into account when performing statistical analyses

2) The choice of analysis (e.g. Frequentist, Bayesian or machine learning) is driven by particular data type

3) Frequentist stats are focused on summary statistics and p-values which can be misleading

4) Bayesian stats deliver uncertainty which facilitates safer predictive analyses

Acknowledgments: LIOS + TARGETWISE

Latvian Institute of Organic Synthesis

2027 National Development Plan

Funded by the European Union