

# SC1\_Proj

Alessio

13 January 2020

## Introduction

One of the most common types of cancer diagnosed in women is breast cancer. There are multiple tests that people are subjected to, but one of the most indicative ones is fine needle aspiration which involves extracting a sample of cells to be examined under a microscope. Multiple numerical metrics are computed from the obtained images. The aim is to use the extracted metrics to make accurate diagnoses.

The dataset consists of 569 images which have been processed as described and a total of 30 variables have been computed for each observation.

The aim of this report is to implement a number of classification algorithms, use them to obtain predictions, and compare their performances.

## Exploratory analysis

```
data <- read_csv("../data/data.csv")
colnames(data)[3:32] <- c('radius_m', 'texture_m', 'perim_m', 'area_m', 'smooth_m', 'compact_m',
                          'concav_m', 'concav_pt_m', 'symmetry_m', 'frac_dim_m', 'radius_se',
                          'texture_se', 'perim_se', 'area_se', 'smooth_se', 'compact_se',
                          'concav_se', 'concav_pt_se', 'symmetry_se', 'frac_dim_se', 'radius_w',
                          'texture_w', 'perim_w', 'area_w', 'smooth_w', 'compact_w', 'concav_w',
                          'concav_pt_w', 'symmetry_w', 'frac_dim_w')
```

Check for missing values in every column.

```
colSums(is.na(data))
```

```
##      id      diagnosis      radius_m      texture_m      perim_m      area_m
##      0          0          0          0          0          0
## smooth_m compact_m      concav_m      concav_pt_m      symmetry_m      frac_dim_m
##      0          0          0          0          0          0
## radius_se texture_se      perim_se      area_se      smooth_se      compact_se
##      0          0          0          0          0          0
## concav_se concav_pt_se      symmetry_se      frac_dim_se      radius_w      texture_w
##      0          0          0          0          0          0
##      perim_w      area_w      smooth_w      compact_w      concav_w      concav_pt_w
##      0          0          0          0          0          0
## symmetry_w      frac_dim_w      X33
##      0          0          569
```

```
data %<>% mutate_at(vars(diagnosis), factor)
```

```
train <- data %>% sample_frac(0.8)
test <- anti_join(data,train, by='id')
```

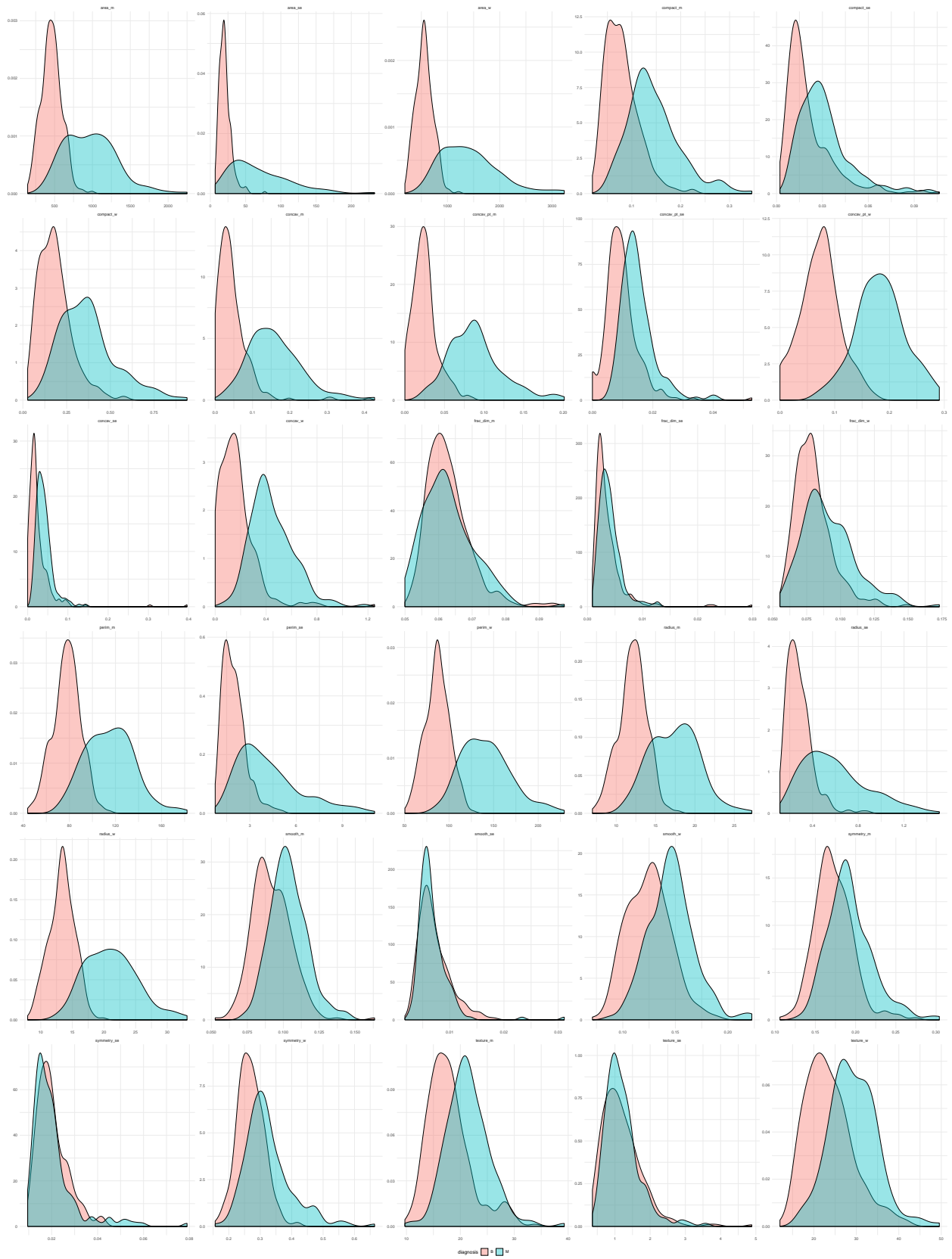
```
# need ids for later
id_train <- train$id
id_test <- test$id
```

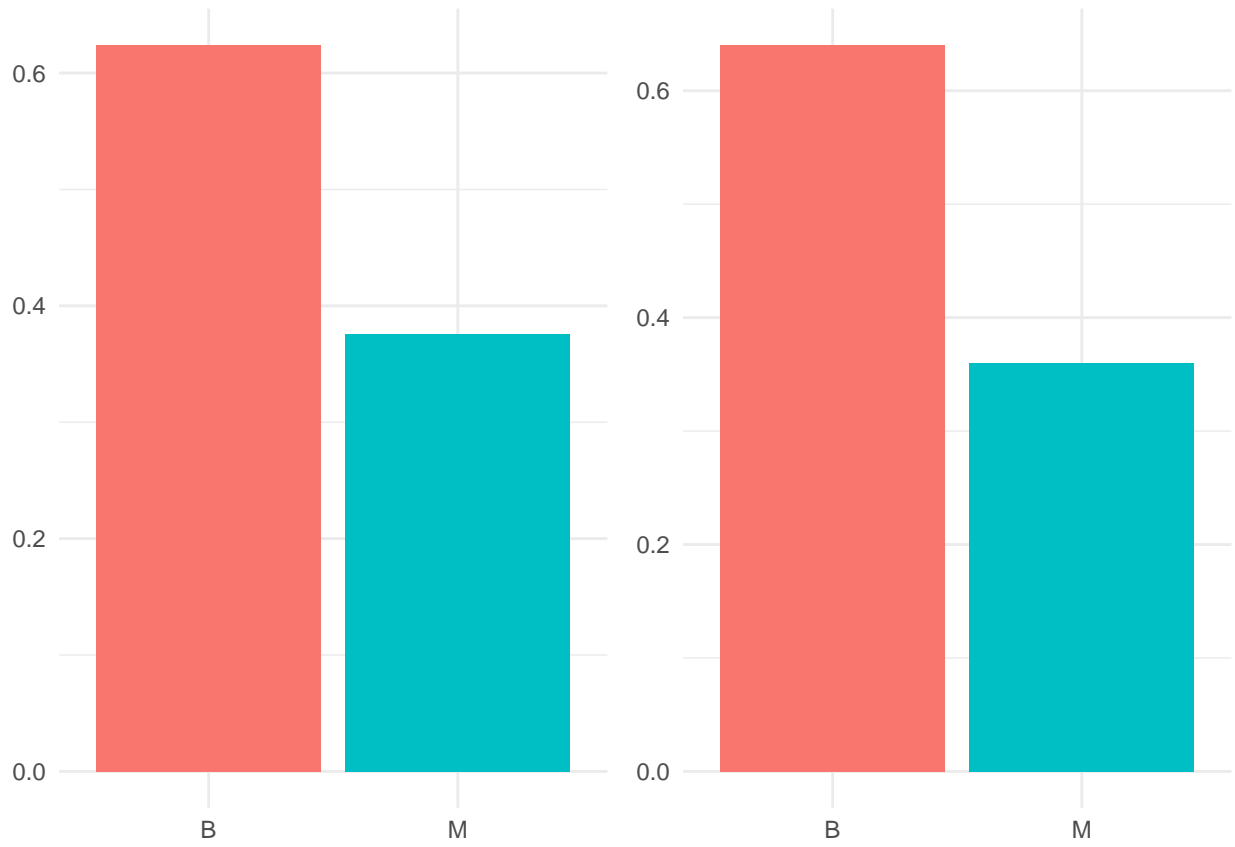
```
data %<>%
  dplyr::select(-c(id, X33))
train %<>%
  dplyr::select(-c(id, X33))
test %<>%
  dplyr::select(-c(id, X33))
```

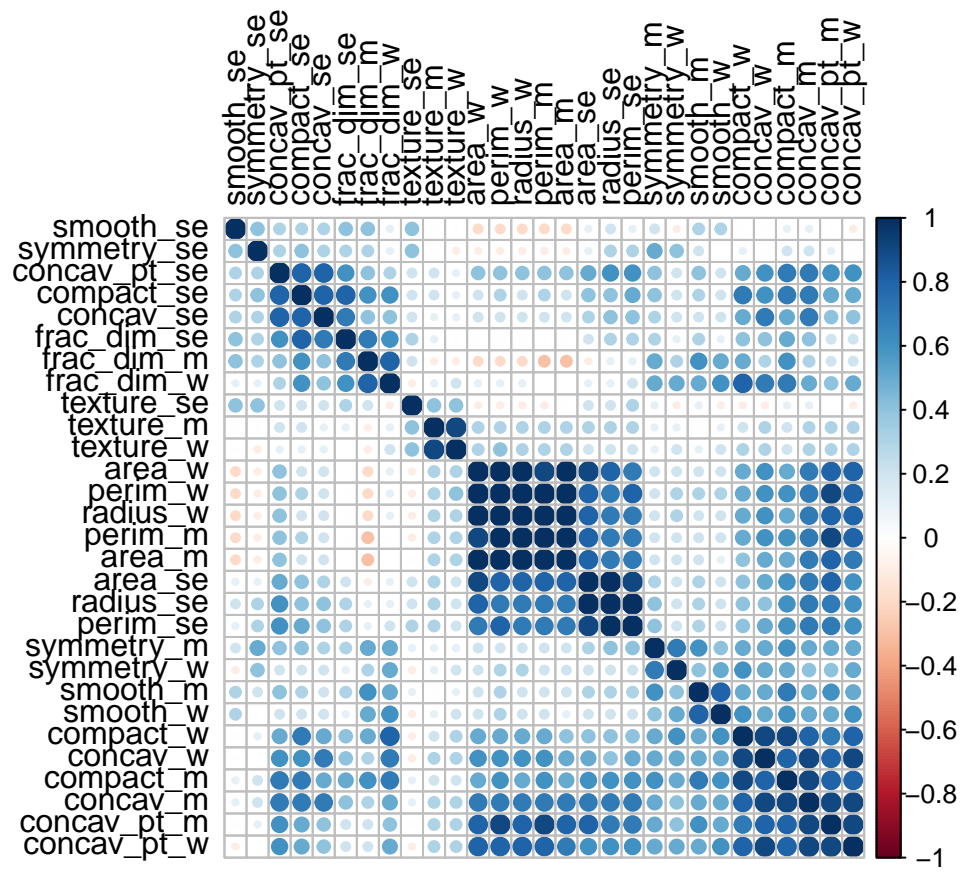
```
sum(is.na(data))
```

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

```
training_data <- train[2:dim(train)[2]]
training_classes <- train[1]
test_data <- test[2:dim(test)[2]]
test_classes <- test[1]
```







# Dimensionality Reduction and Feature Selection

## PCA

Code

```
normalise_z <- function(X){
  mean_cols <- colMeans(X)
  sd_cols <- apply(X, 2, sd)
  mean_normalised_X <- t(apply(X, 1, function(x){x - mean_cols}))
  normalised_X <- t(apply(mean_normalised_X, 1, function(x){x / sd_cols}))
  return(normalised_X)
}

pca <- function(X, number_components_keep) {
  normalised_X <- normalise_z(X)

  corr_mat <- t(normalised_X) %*% normalised_X

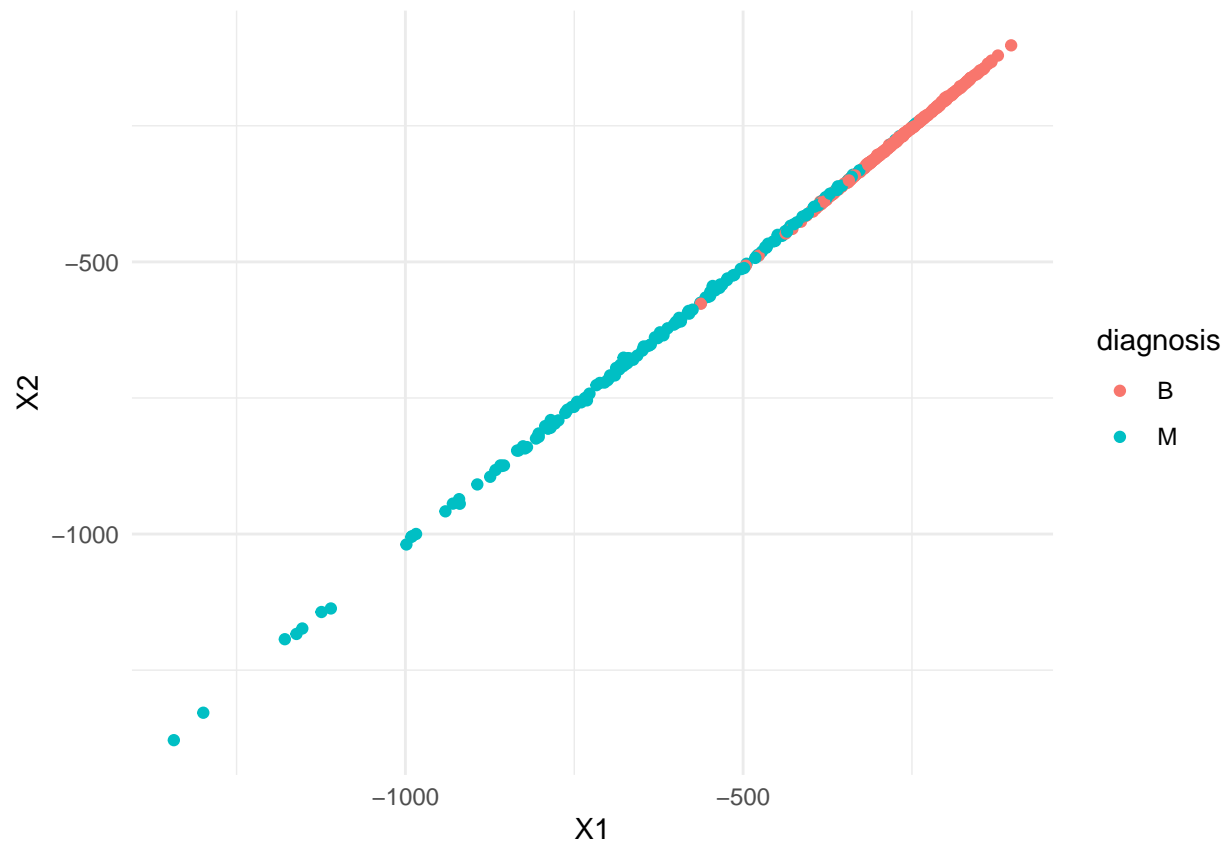
  eigenvectors <- eigen(corr_mat, symmetric=TRUE)$vectors

  reduced_data <- X %*% eigenvectors[,1:number_components_keep]
  relevant_eigs <- eigenvectors[,1:number_components_keep]
  returnds <- list(reduced_data, relevant_eigs)
  names(returnds) <- c("reduced_data", "reduction_matrix")
  return(returnds)
}
```

Apply to dataset

```
pca_result <- pca(as.matrix(training_data), 2)
pca_reduced_training_data <- data.frame(cbind(pca_result$reduced_data, training_classes))

ggplot(data=pca_reduced_training_data, aes(x=X1, y=X2)) + geom_point(aes(colour=diagnosis))
```



- Correlation Feature Selection
- LDA

## Classification

add list of methods+measure

## SVM

To solve the problem of finding a SVM like classifier for non-separable data we must permit a certain number of points to violate the boundaries set however this number and the amount they violate the constraints by must be as small as possible. To formulate this we introduce a variable  $\epsilon_i$  for each data point into the objective functions and the constraints leading to the optimisation problem:

$$\min_{w, \epsilon_i} \frac{1}{2} w^T w + C \sum_{i=0}^n \epsilon_i$$

such that  $w \cdot x_i + b + \epsilon_i > 1$  if  $y_i = 1$   
and  $w \cdot x_i + b + \epsilon_i < -1$  if  $y_i = -1$

Note that we have swapped the sign of the  $b$  term in the equation for the hyperplane because I implemented it this way before realising they were different and am lazy.

As the above problem is convex (as it is quadratic) and Slater's condition holds then strong duality holds and we can take the Lagrangian of the optimisation problem and consider the result of the KKT conditions. By doing so we can reformulate the optimisation problem as the dual problem:

$$\min_{\lambda} \frac{\bar{\lambda} X X^T \bar{\lambda}^T}{4} + \lambda^T \mathbf{1}$$

such that  $0 \leq \lambda_i \leq C$   
and  $\sum_i \lambda_i y_i = 0$

where

$$X = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \text{ and } \bar{\lambda} = [\lambda_1 \cdot y_1, \dots, \lambda_n \cdot y_n] \text{ and } \mathbf{1} = [1, \dots, 1] \in \mathbb{R}^n$$

As before we have to massage this optimisation problem into one that can be solved using `solve.QP`. In this formulation

$$d = \mathbf{1}$$

and

$$D = \begin{pmatrix} y_1 & 0 & \dots & 0 \\ 0 & y_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & y_n \end{pmatrix} X X^T \begin{pmatrix} y_1 & 0 & \dots & 0 \\ 0 & y_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & y_n \end{pmatrix}$$

$A$  and  $b_0$  require slightly more manipulation this time around with

$$A = \begin{pmatrix} y_1 & y_2 & \dots & y_n \\ & I & & \\ & & -I & \end{pmatrix}$$



and

$$b_0 = \begin{pmatrix} 0 \\ \mathbf{0} \\ -C \end{pmatrix}$$

where

$$\mathbf{0} = [0, \dots, 0]^T \in \mathbb{R}^n$$

and

$$C = [C, \dots, C]^T \in \mathbb{R}^n$$

The code for this applied to the non-separable data can be found below.

```
C <- 1

X <- as.matrix(combined_class)[,1:2]
y <- as.matrix(combined_class)[,3]
Dmat2 <- diag(y) * X %*% t(X) %*% diag(y)
diag(Dmat2) <- diag(Dmat2) + 1e-11
dv2 <- rep(1, 30)

A2 <- rbind( y,diag(30))
A2 <- rbind(A2, -1*diag(30))

bv2 <- c(c(0), rep(0, 30), rep(-C, 30) )
model <- solve.QP(Dmat2, dv2, t(A2), bv2, meq = 1)
```

In order to recover  $w$  and  $b$  from  $\lambda$  we use the relationship

$$w = \sum_{i=0}^{n-1} \lambda_i x_i^T y_i$$

and

$$b = \text{mean}\left(\sum_{i=0}^k y_i - w \cdot x_i\right) \cdot \forall i. 0 < \lambda_i < C$$

Which can be made as functions in R as so:

```
calculate_b <- function(w, X, y, a, C) {
  ks <- sapply(a, function(x){return(x > 0 && x < C)})
  indices <- which(ks)
  sum_bs <- 0
  for(i in indices) {
    sum_bs <- sum_bs + (y[i] - w %*% X[i,])
  }
  return(sum_bs / length(indices))
}

recover_w <- function(a, y, X){
  colSums(diag(a) %*% diag(y) %*% X)
}
```

We can see the results of using the dual regression below

```

soft_margin_svm_plotter <- function(w, b) {
  plotter <- function(x) {
    return(1/w[2] * -(b + (w[1]*as.numeric(x))))
  }
  return(plotter)
}

```

```

factor_to_label <- function(x) {
  if(as.character(x) == "M") {
    return(1)
  }
  else {
    return(-1)
  }
}

```

```

label_to_factor <- function(x) {
  if(x == 1) {
    return(as.factor("M"))
  }
  else{
    return(as.factor("B"))
  }
}
numeric_test_labels <- apply(test_classes, 1, factor_to_label)
numeric_training_labels <- apply(training_classes, 1, factor_to_label)

```

Use PCA then do SVM

```

model <- svm(X=pca_result$reduced_data,
             classes=numeric_training_labels,
             C=100000, margin_type='soft',
             kernel_function = linear_kernel,
             feature_map = linear_basis_function)

```

```

reduced_prediction_fn <- model$prediction_function

```

```

pca_reduced_prediction_fn <- function(x) {
  p <- x %*% pca_result$reduction_matrix
  reduced_prediction_fn(t(p))
}

```

```

pred_svm <- apply(as.matrix(test_data),1, pca_reduced_prediction_fn)
accuracy_calc(numeric_test_labels, pred_svm)

```

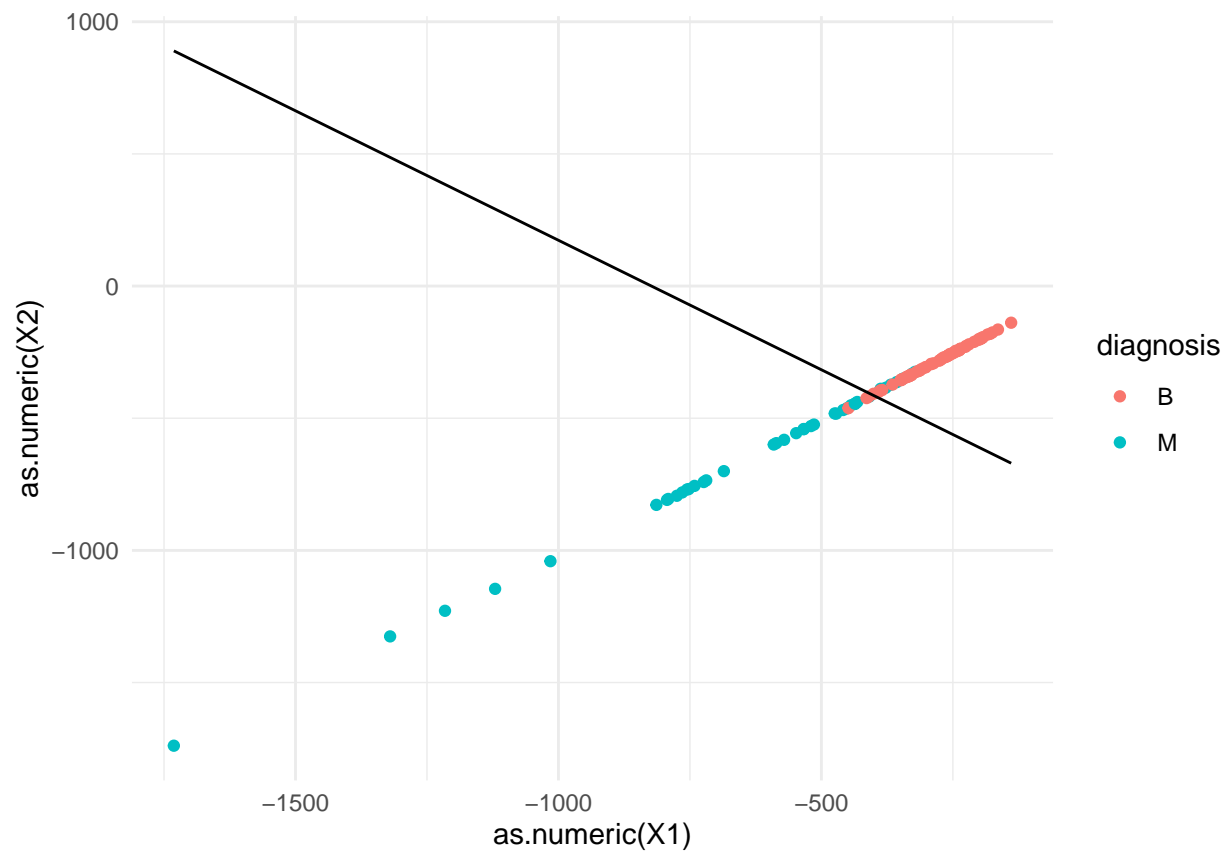
```
## [1] 89.47368
```

```

svm_plotter <- soft_margin_svm_plotter(model$params$w, model$params$b)
embedded_test_data <- data.frame(cbind(as.matrix(test_data) %*% pca_result$reduction_matrix), test_classes)

```

```
ggplot(embedded_test_data, aes(x=as.numeric(X1), y=as.numeric(X2))) +
  geom_point(aes(colour=diagnosis)) +
  stat_function(fun=svm_plotter)
```



# Naive Bayes

## Mathematical setting

Let  $y$  be the class label that we want to assign to an observation  $\mathbf{x} = (x_1, \dots, x_d)$ , where  $x_1, \dots, x_d$  are the features. The probability of an observation having label  $y$  is given by Bayes rule,

$$P(y|x_1, \dots, x_d) = \frac{P(x_1, \dots, x_d|y)P(y)}{P(x_1, \dots, x_d)} \\ \propto P(x_1, \dots, x_d|y)P(y).$$

The prior class probability  $P(y)$  can be easily obtained by the proportion of observation that are in the given class.

The main assumption is that every feature is conditionally independent given the class label  $y$ . The reason why this classifier is called *naïve* is that very often this assumption is not actually realistic.

This assumption simplifies the posterior to

$$P(y|x_1, \dots, x_d) \propto P(y) \prod_{i=1}^d P(x_i|y).$$

There are various types of Naive Bayes classifiers based on the type of features. In our case, since we have continuous variables we assume that all features are normally distributed. Therefore, the conditional probabilities can be calculated as

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Finally, to assign the class to an observation we use the Maximum A Posteriori decision rule. For every observation, we pick the class the has the highest probability

$$y = \underset{y}{\operatorname{argmax}} P(y) \prod_{i=1}^d P(x_i|y).$$

## Implementation

*Here are some code snippets just to illustrate how these theoretical aspects are implemented. The full code can be found in the package.*

The observations are stored as rows in  $X$  and the corresponding class labels are entires in the column matrix  $y$ .

First we calculate the prior class probabilities based on the number of observations in each class.

```
n <- dim(X)[1]
d <- dim(X)[2]
classes <- sort(unique(y)[, 1])
k <- length(classes)

prior <- rep(0, k)
for (i in 1:k) {
  prior[i] <- sum(y == classes[i]) / n
}
```

Then we create an array of the mean and sd of the data split by classes and features.

```
summaries <- array(rep(1, d * k * 2), dim = c(k, d, 2))
for (i in 1:k) {
  X_k <- X[which(y == (i - 1)), ]
  summaries[i, , 1] <- apply(X_k, 2, mean)
  summaries[i, , 2] <- apply(X_k, 2, sd)
}
```

Finally, the predictions are obtained by taking the largest posterior class probability. Note that in order to avoid underflow, we take the maximum of the *log* posterior class probabilities.

```
probs <- matrix(rep(0, n * k), nrow = n)
for (obs in 1:n) {
  for (class in 1:k) {
    class_prob <- log(prior[class])
    for (feat in 1:d) {
      mu <- summaries[class, feat, 1]
      sd <- summaries[class, feat, 2]
      cond <- dnorm(x_new[obs, feat], mu, sd, log = TRUE)
      class_prob <- class_prob + cond
    }
    probs[obs, class] <- class_prob
  }
}

pred <- apply(probs, 1, which.max)
```

## Fit model to dataset

```
levels(training_classes$diagnosis) <- c(0,1)
training_classes %<>% as.matrix
mode(training_classes) <- 'numeric'

levels(test_classes$diagnosis) <-c(0,1)
test_classes %<>% as.matrix
mode(test_classes) <- 'numeric'
```

Fit the Naive Bayes model to the data, calculate predictions and check the accuracy using.

```
model_naive <- naive_bayes(training_data, training_classes)

pred_naive <- predict(model_naive, as.matrix(test_data))

# confusion_plot(test_classes, pred_naive)

calc_accuracy <- function(ytest, yhat) sum(drop(yhat) == drop(ytest)) / length(drop(ytest))

acc_naive <- calc_accuracy(test_classes, pred_naive)
acc_naive

## [1] 0.9210526
```

# Logistic Regression

## Mathematical Setting

Let  $Y_i \mid \mathbf{x}_i \sim \text{Bernoulli}(p_i)$  with  $p_i = \sigma(\mathbf{x}_i^\top \boldsymbol{\beta})$  where  $\sigma(\cdot)$  is the **sigmoid function**. The joint log-likelihood is given by

$$\ln p(\mathbf{y} \mid \boldsymbol{\beta}) = \sum_{i=1}^n y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i) = - \sum_{i=1}^n \ln(1 + \exp((1 - 2y_i)\mathbf{x}_i^\top \boldsymbol{\beta}))$$

## Maximum Likelihood Estimation

Maximizing the likelihood is equivalent to minimizing the negative log-likelihood. Minimizing the negative log likelihood is equivalent to solving the following optimization problem

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^n \ln(1 + \exp((1 - 2y_i)\mathbf{x}_i^\top \boldsymbol{\beta}))$$

## Maximum-A-Posteriori and Ridge Regularization

We can introduce an isotropic Gaussian prior on **all** the coefficients  $p(\boldsymbol{\beta}) = N(\mathbf{0}, \sigma_{\boldsymbol{\beta}}^2 I)$ . Maximizing the posterior  $p(\boldsymbol{\beta} \mid \mathbf{y})$  is equivalent to minimizing the negative log posterior  $-\ln p(\boldsymbol{\beta} \mid \mathbf{y})$  giving

$$\min_{\boldsymbol{\beta}} \sigma_{\boldsymbol{\beta}}^2 \sum_{i=1}^n \ln(1 + \exp((1 - 2y_i)\mathbf{x}_i^\top \boldsymbol{\beta})) + \frac{1}{2} \boldsymbol{\beta}^\top \boldsymbol{\beta}$$

Often we don't want to regularize the intercept, future work could look into placing an isotropic Gaussian prior on  $\boldsymbol{\beta}_{1:p-1} := (\beta_1, \dots, \beta_{p-1})$  and instead we place a uniform distribution on  $\beta_0$ , which doesn't depend on  $\beta_0$ . This would leads to

$$\min_{\boldsymbol{\beta}} \sigma_{\boldsymbol{\beta}_{1:p-1}}^2 \sum_{i=1}^n \ln(1 + \exp((1 - 2y_i)\mathbf{x}_i^\top \boldsymbol{\beta})) + \frac{1}{2} \boldsymbol{\beta}_{1:p-1}^\top \boldsymbol{\beta}_{1:p-1}$$

## Gradient Ascent (MLE, No Regularization)

Updates take the form

$$\boldsymbol{\beta}_{k+1} \leftarrow \boldsymbol{\beta}_k + \gamma X^\top (\mathbf{y} - \sigma(X\boldsymbol{\beta}_k))$$

## Maximum-A-Posteriori with BFGS, Newton's Method and Gradient Ascent

```
Xtrain <- as.matrix(cbind(1, training_data))
Xtest  <- as.matrix(cbind(1, test_data))
# MAP, BFGS
lr_map_bfgs <- logistic_regression(Xtrain, training_classes, cost="MAP", method="BFGS")
yhat_map_bfgs <- predict(lr_map_bfgs, Xtest)
acc_map_bfgs <- calc_accuracy(test_classes, yhat_map_bfgs)
# MAP, NEWTON
lr_map_nm <- logistic_regression(Xtrain, training_classes, cost="MAP", method="NEWTON", niter=250)
yhat_map_nm <- predict(lr_map_nm, Xtest)
```

```

acc_map_nm <- calc_accuracy(test_classes, yhat_map_nm)
# MAP, GRADIENT ASCENT
lr_map_ga <- logistic_regression(Xtrain, training_classes, cost="MAP", method="GA", niter=1000)
yhat_map_ga <- predict(lr_map_ga, Xtest)
acc_map_ga <- calc_accuracy(test_classes, yhat_map_ga)

```

## Maximum Likelihood Estimation with BFGS, Newton's Method and Gradient Ascent

```

# MLE, BFGS
lr_mle_bfgs <- logistic_regression(Xtrain, training_classes, cost="MLE", method="BFGS")
yhat_mle_bfgs <- predict(lr_mle_bfgs, Xtest)
acc_mle_bfgs <- calc_accuracy(test_classes, yhat_mle_bfgs)
# MLE, NEWTON
lr_mle_nm <- logistic_regression(Xtrain, training_classes, cost="MLE", method="NEWTON", niter=250)
yhat_mle_nm <- predict(lr_mle_nm, Xtest)
acc_mle_nm <- calc_accuracy(test_classes, yhat_mle_nm)
# MLE, GRADIENT ASCENT
lr_mle_ga <- logistic_regression(Xtrain, training_classes, cost="MLE", method="GA", niter=1000)
yhat_mle_ga <- predict(lr_mle_ga, Xtest)
acc_mle_ga <- calc_accuracy(test_classes, yhat_mle_ga)

```

## Combine results in a single matrix

```

# put everything together into a nice table
results_matrix <- matrix(c(acc_mle_bfgs, acc_mle_nm, acc_mle_ga,
                           acc_map_bfgs, acc_map_nm, acc_map_ga),
                         dimnames=list(c("BFGS", "NM", "GA"), c("MLE", "MAP")),
                         nrow=3, ncol=2)
results <- data.frame(results_matrix)
results

```

```

##           MLE           MAP
## BFGS 0.9385965 0.9561404
## NM   0.9385965 0.9561404
## GA   0.8859649 0.8947368

```

## Random-Walk Metropolis-Hastings

```

rwmh_multivariate_log <- function(start, niter, logtarget, vcov, thinning, burnin){
  # Set current z to the initial point and calculate its log target to save computations
  z <- start # It's a column vector
  pz <- logtarget(start)
  # create vector deciding iterations where we record the samples
  store <- seq(from=(1+burnin), to=niter, by=thinning)
  #n_samples <- (niter - burnin) %/% thinning
  # Generate matrix containing the samples. Initialize first sample with the starting value
  samples <- matrix(0, nrow=length(store), ncol=nrow(start))

```

```

samples[1, ] <- start
# Generate uniform random numbers in advance, to save computation. Take logarithm?
log_u <- log(runif(niter))
# Proposal is a multivariate standard normal distribution. Generate samples and
# later on use linearity property of Gaussian distribution
vcov <- diag(nrow(start)) %*% vcov
normal_shift <- mvrnorm(n=niter, mu=c(0,0,0), Sigma=vcov)
for (i in 2:niter){
  # Sample a candidate
  candidate <- z + normal_shift[i, ]
  # calculate log target of candidate and store it in case it gets accepted
  p_candidate <- logtarget(candidate)
  # use decision rule explained in blog posts
  if (log_u[i] <= p_candidate - pz){
    # Accept!
    z <- candidate
    pz <- p_candidate
  }
  # Finally add the sample to our matrix of samples
  if (i %in% store) samples[which(store==i), ] <- z
}
return(samples)
}

```

Log posterior using the 2 dimensional data

```

# add a column of 1 for bias
Xrwmh <- cbind(1, as.matrix(pca_reduced_training_data[, c(1, 2)]))
# up to normalizing constant
log_posterior_unnormalized <- function(beta){
  log_prior <- -0.5*sum(beta^2)
  log_likelihood <- -sum(log(1 + exp((1 - 2*training_classes) * (Xrwmh %*% beta))))
  return(log_prior + log_likelihood)
}

```

Start Optimization at the Mode

```

# To start the algorithm more efficiently, start from MAP estimate
optim_results <- optim(c(0,0,0), function(x) - log_posterior_unnormalized(x), method="BFGS", hessian=TRUE)
start <- matrix(optim_results$par)
# Use inverse of approximate hessian matrix as vcov of normal proposal
vcov <- solve(optim_results$hessian)

```

Set settings

```

niter <- 50000
thinning <- 1

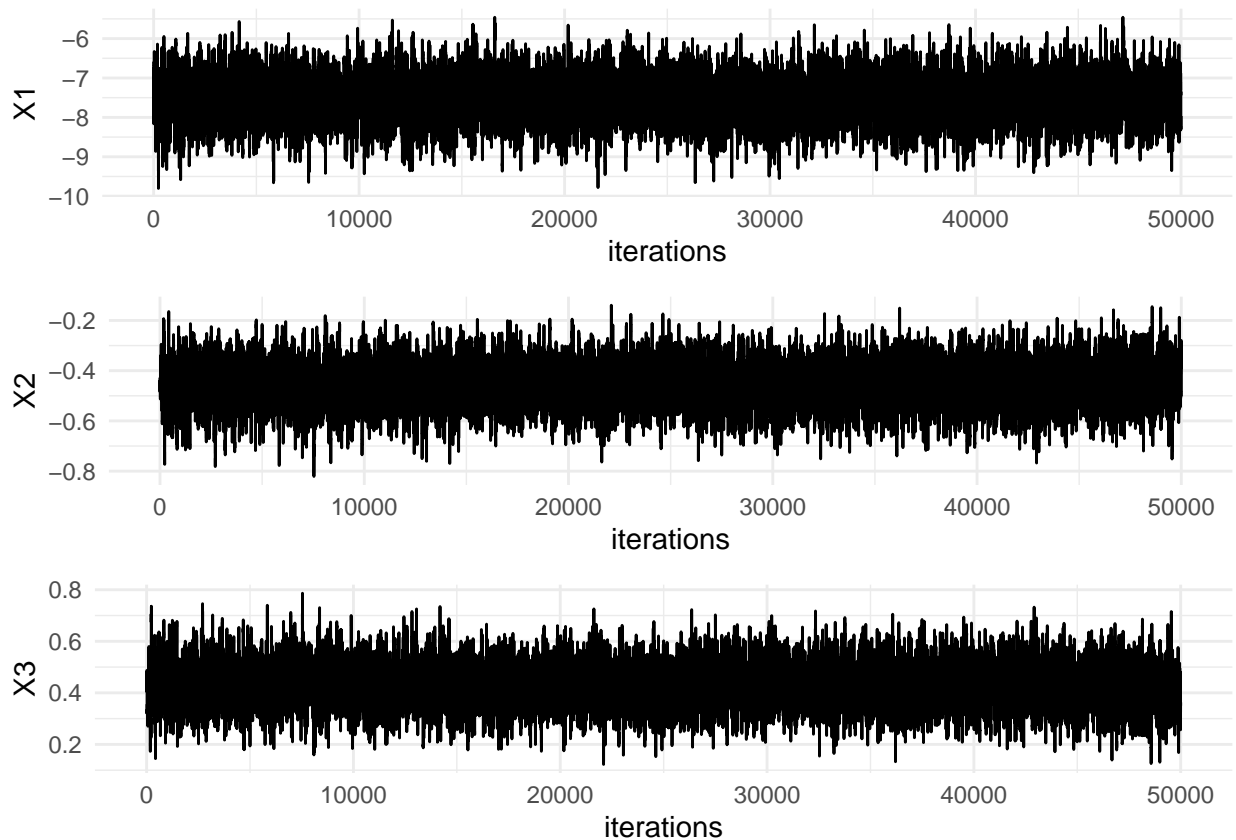
```



```
burnin <- 0
samples <- rwmh_multivariate_log(start, niter, log_posterior_unnormalized, vcov, thinning, burnin)
```

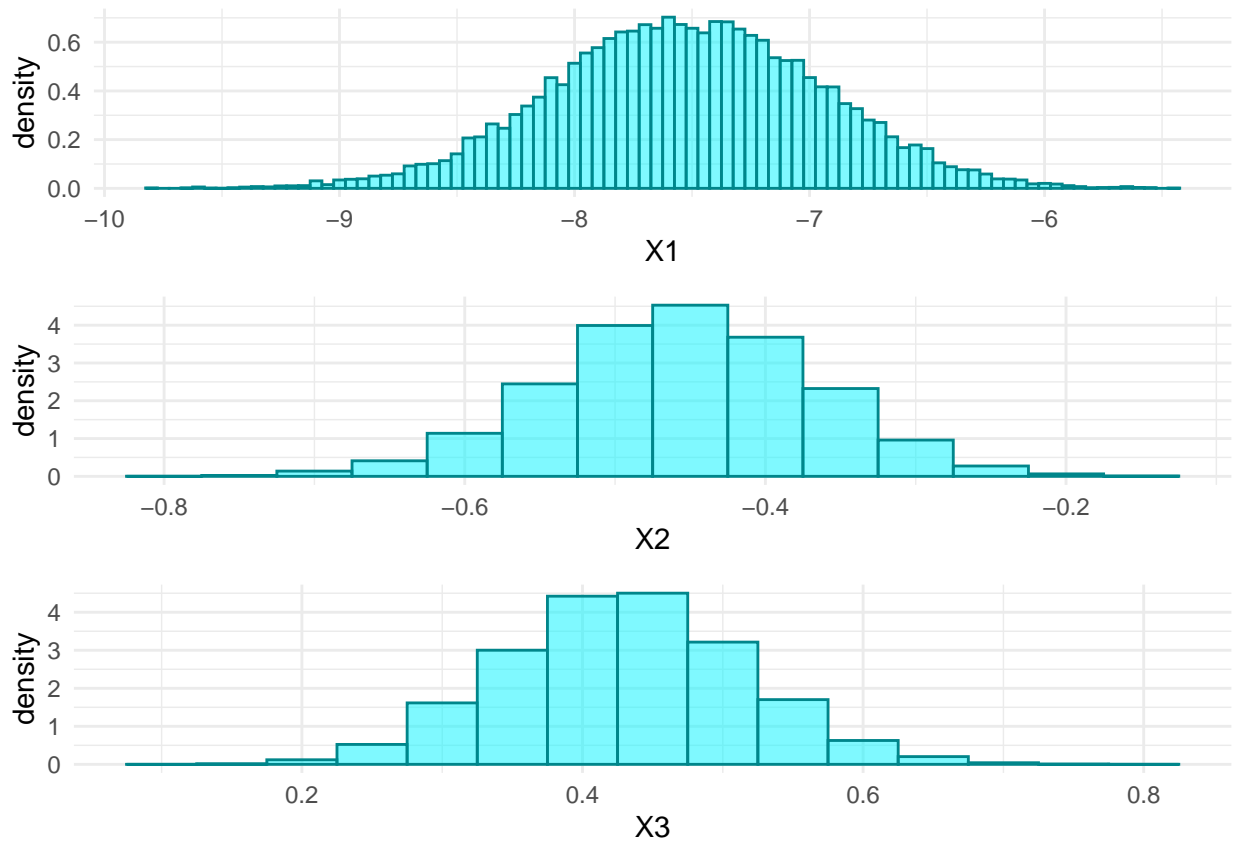
## Plot samples

```
samplesdf <- data.frame(samples) %>% mutate(iterations=row_number())
trace1 <- ggplot(data=samplesdf, aes(x=iterations, y=X1)) + geom_line()
trace2 <- ggplot(data=samplesdf, aes(x=iterations, y=X2)) + geom_line()
trace3 <- ggplot(data=samplesdf, aes(x=iterations, y=X3)) + geom_line()
grid.arrange(trace1, trace2, trace3, ncol=1)
```



## Histogram of Samples

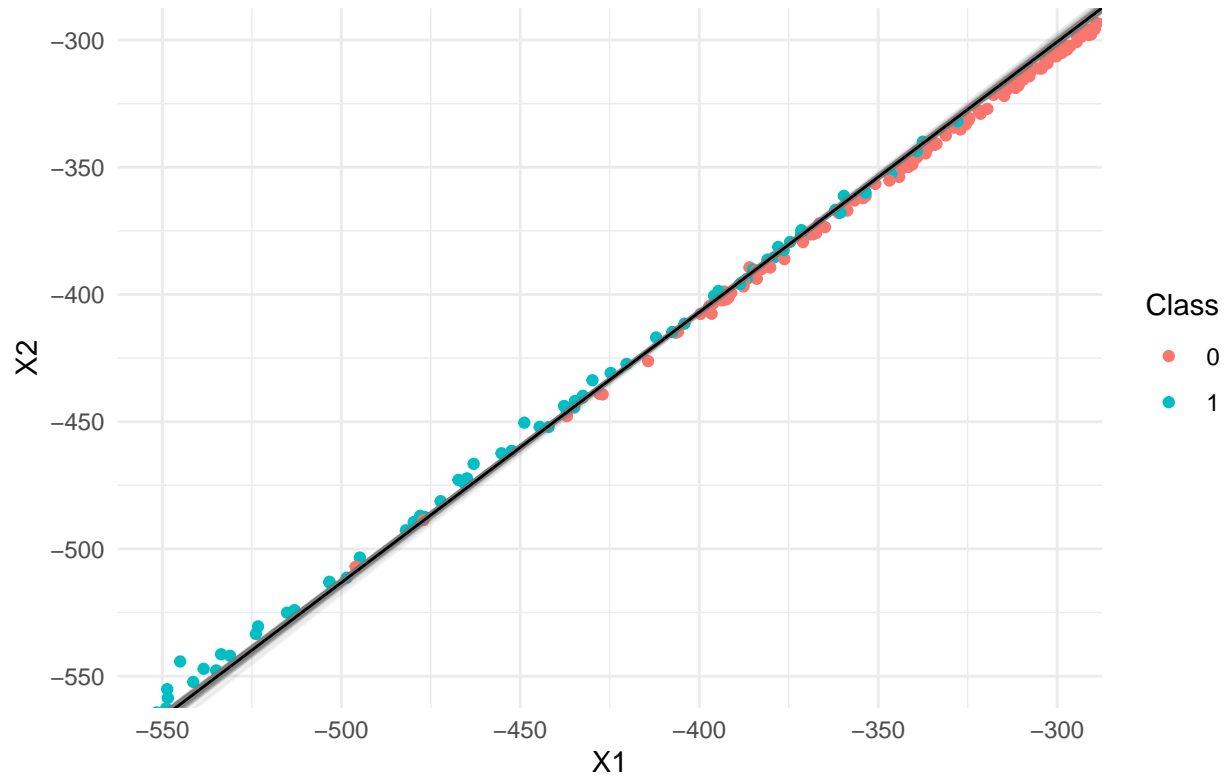
```
hist1 <- ggplot(data=samplesdf, aes(x=X1, stat(density))) +
  geom_histogram(binwidth=0.05, alpha=0.5, fill="turquoise1", color="turquoise4")
hist2 <- ggplot(data=samplesdf, aes(x=X2, stat(density))) +
  geom_histogram(binwidth=0.05, alpha=0.5, fill="turquoise1", color="turquoise4")
hist3 <- ggplot(data=samplesdf, aes(x=X3, stat(density))) +
  geom_histogram(binwidth=0.05, alpha=0.5, fill="turquoise1", color="turquoise4")
grid.arrange(hist1, hist2, hist3, ncol=1)
```



plot line I guess

```
samples_to_select <- 200
samples_subset <- samples[sample(1:nrow(samples), samples_to_select), ]
# Calculate slope and intercept for those samples. Then calculate x2 from x1
linecoefs <- cbind(-samples_subset[, 2]/samples_subset[, 3], - samples_subset[, 1] / samples_subset[, 3])
x2_vals <- apply(linecoefs, 1, function(row) Xrwmh[, 2]*row[1] + row[2])
# Store x2 values together with x1 values from X. Then melt to plot all lines
dfsampl_lines <- data.frame(x1=Xrwmh[, 2], x2_vals) %>%
  gather("key", "value", -x1)
# replace diagnosis
data_plot <- pca_reduced_training_data %>% mutate(training_classes=training_classes)
# Create dataframe containing values for the MAP line
dfmap <- data.frame(x1=Xrwmh[, 2], y=(-start[2, ]/start[3, ])*Xrwmh[, 2] + (-start[1, ]/start[3, ]))
ggplot() +
  geom_point(data=data_plot, aes(x=X1, y=X2, color=as.factor(training_classes))) + # Dataset scatter
  coord_cartesian(xlim=c(-550, -300), ylim=c(-550, -300)) +
  geom_line(data=dfsampl_lines, aes(x=x1, y=value, group=key), alpha=0.1, color="grey50") +
  geom_line(data=dfmap, aes(x=x1, y=y), color="black") + # MAP line
  labs(color="Class", title="Sample Decision Boundaries") +
  theme(plot.title=element_text(hjust=0.5, size=20))
```

## Sample Decision Boundaries



## Conclusion

Merge all predictions

```
id <- seq(length(id_test))
all_pred <- cbind(id,numeric_test_labels,pred_naive,pred_svm)
colnames(all_pred) <- c('id','actual','naive','svm')
all_pred[all_pred== -1] <- 0
all_pred %<>% as.data.frame()
```

```
confusion_plot <- function(actual,predicted){
  confusion_matrix <- as.data.frame(table(actual,predicted))
  g <-ggplot(confusion_matrix,aes(x=actual,y=predicted))+
    geom_tile(aes(fill=Freq))+
    geom_text(aes(label=sprintf("%1.0f", Freq)),color="white",fontface="bold")+
    labs(x="Actual class",y="Predicted class")+
    theme_minimal()
  return(g)
}
```

```
errors <- all_pred %>%
  mutate(naive=naive==actual,
         svm=svm==actual,
         logistic=yhat_map_bfgs==actual) %>%
  dplyr::select(-'actual') %>%
  melt(id='id')
```

Visualize which are the observations that the models missclassify.

```
ggplot(errors,aes(x=id,y=variable,fill=value))+
  geom_raster()+
  theme_minimal()
```

