SC1_Proj

Alessio

13 January 2020

Dataset

The dataset is TODO: describe

```
data <- read_csv("../data/data.csv")
glimpse(data)
```

```
## Observations: 569
## Variables: 33
## $ id                      <dbl> 842302, 842517, 84300903, 84348301, 8435840...
## $ diagnosis               <chr> "M", "M", "M", "M", "M", "M", "M", "M", "M"...
## $ radius_mean             <dbl> 17.990, 20.570, 19.690, 11.420, 20.290, 12....
## $ texture_mean            <dbl> 10.38, 17.77, 21.25, 20.38, 14.34, 15.70, 1...
## $ perimeter_mean          <dbl> 122.80, 132.90, 130.00, 77.58, 135.10, 82.5...
## $ area_mean               <dbl> 1001.0, 1326.0, 1203.0, 386.1, 1297.0, 477....
## $ smoothness_mean         <dbl> 0.11840, 0.08474, 0.10960, 0.14250, 0.10030...
## $ compactness_mean        <dbl> 0.27760, 0.07864, 0.15990, 0.28390, 0.13280...
## $ concavity_mean          <dbl> 0.30010, 0.08690, 0.19740, 0.24140, 0.19800...
## $ `concave points_mean`   <dbl> 0.14710, 0.07017, 0.12790, 0.10520, 0.10430...
## $ symmetry_mean           <dbl> 0.2419, 0.1812, 0.2069, 0.2597, 0.1809, 0.2...
## $ fractal_dimension_mean  <dbl> 0.07871, 0.05667, 0.05999, 0.09744, 0.05883...
## $ radius_se               <dbl> 1.0950, 0.5435, 0.7456, 0.4956, 0.7572, 0.3...
## $ texture_se              <dbl> 0.9053, 0.7339, 0.7869, 1.1560, 0.7813, 0.8...
## $ perimeter_se            <dbl> 8.589, 3.398, 4.585, 3.445, 5.438, 2.217, 3...
## $ area_se                 <dbl> 153.40, 74.08, 94.03, 27.23, 94.44, 27.19, ...
## $ smoothness_se           <dbl> 0.006399, 0.005225, 0.006150, 0.009110, 0.0...
## $ compactness_se          <dbl> 0.049040, 0.013080, 0.040060, 0.074580, 0.0...
## $ concavity_se            <dbl> 0.05373, 0.01860, 0.03832, 0.05661, 0.05688...
## $ `concave points_se`     <dbl> 0.015870, 0.013400, 0.020580, 0.018670, 0.0...
## $ symmetry_se             <dbl> 0.03003, 0.01389, 0.02250, 0.05963, 0.01756...
## $ fractal_dimension_se    <dbl> 0.006193, 0.003532, 0.004571, 0.009208, 0.0...
## $ radius_worst            <dbl> 25.38, 24.99, 23.57, 14.91, 22.54, 15.47, 2...
## $ texture_worst           <dbl> 17.33, 23.41, 25.53, 26.50, 16.67, 23.75, 2...
## $ perimeter_worst         <dbl> 184.60, 158.80, 152.50, 98.87, 152.20, 103....
## $ area_worst              <dbl> 2019.0, 1956.0, 1709.0, 567.7, 1575.0, 741....
## $ smoothness_worst        <dbl> 0.1622, 0.1238, 0.1444, 0.2098, 0.1374, 0.1...
## $ compactness_worst       <dbl> 0.6656, 0.1866, 0.4245, 0.8663, 0.2050, 0.5...
## $ concavity_worst         <dbl> 0.71190, 0.24160, 0.45040, 0.68690, 0.40000...
## $ `concave points_worst`  <dbl> 0.26540, 0.18600, 0.24300, 0.25750, 0.16250...
## $ symmetry_worst          <dbl> 0.4601, 0.2750, 0.3613, 0.6638, 0.2364, 0.3...
## $ fractal_dimension_worst <dbl> 0.11890, 0.08902, 0.08758, 0.17300, 0.07678...
## $ X33                     <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
```

```
colnames(data)[3:32] <- c('radius_m','texture_m', 'perim_m','area_m','smooth_m','compact_m','concav_m',
```

Dataset Preprocessing Visualisation and Exploration

```
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)
```

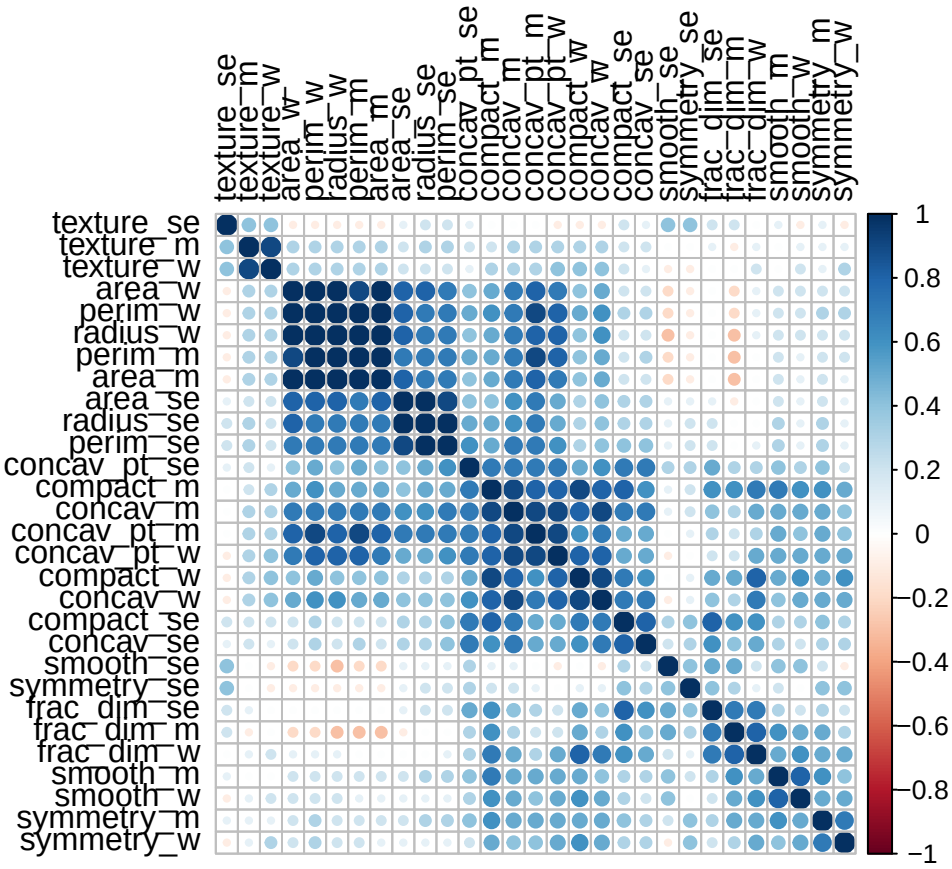
```
train <- data %>% sample_frac(0.8)
test <- anti_join(data,train, by='id')

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

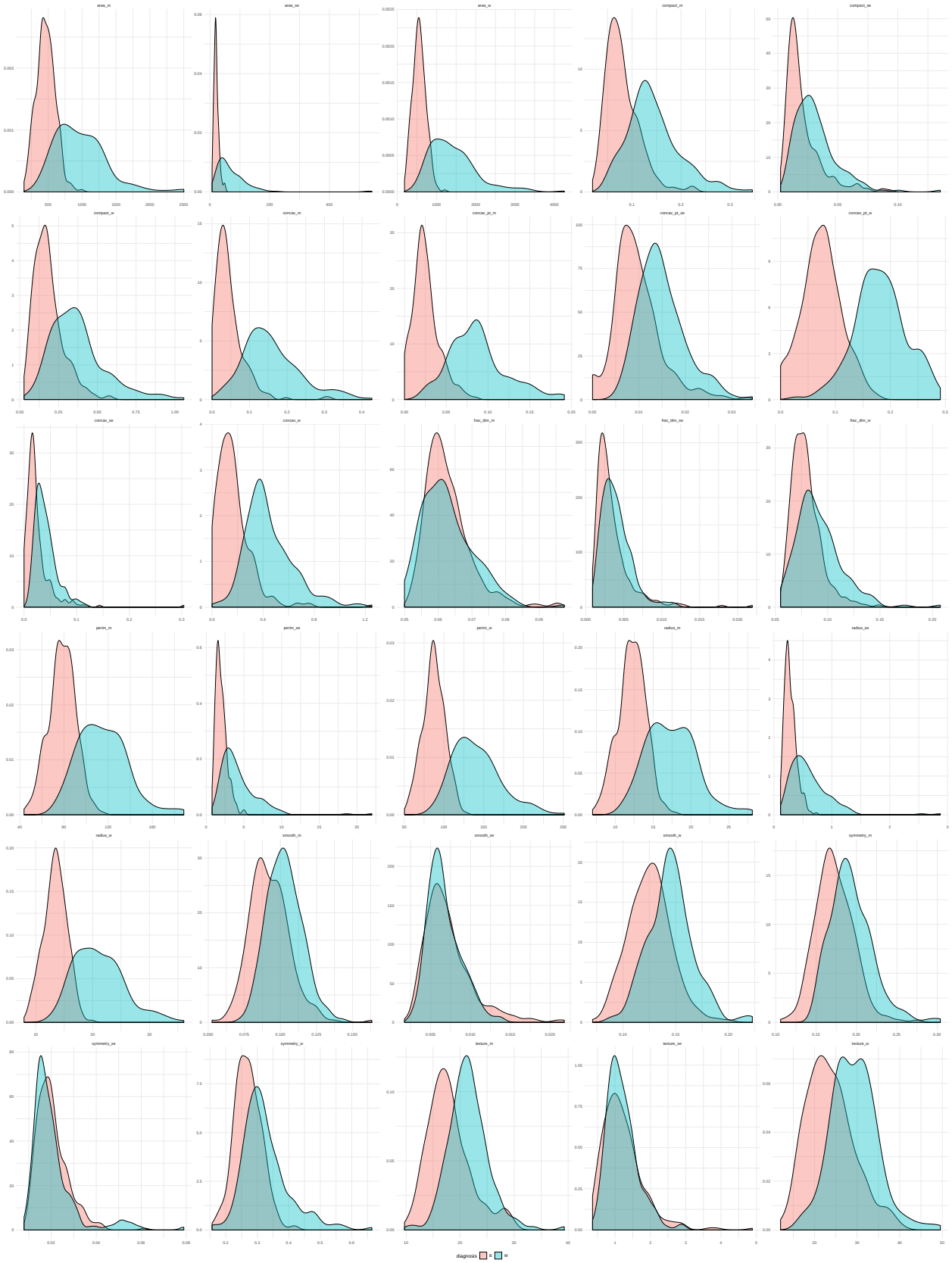
```
data %<>%
  dplyr::select(-c(id, X33))
train %<>%
  dplyr::select(-c(id, X33))
test %<>%
  dplyr::select(-c(id, X33))
```

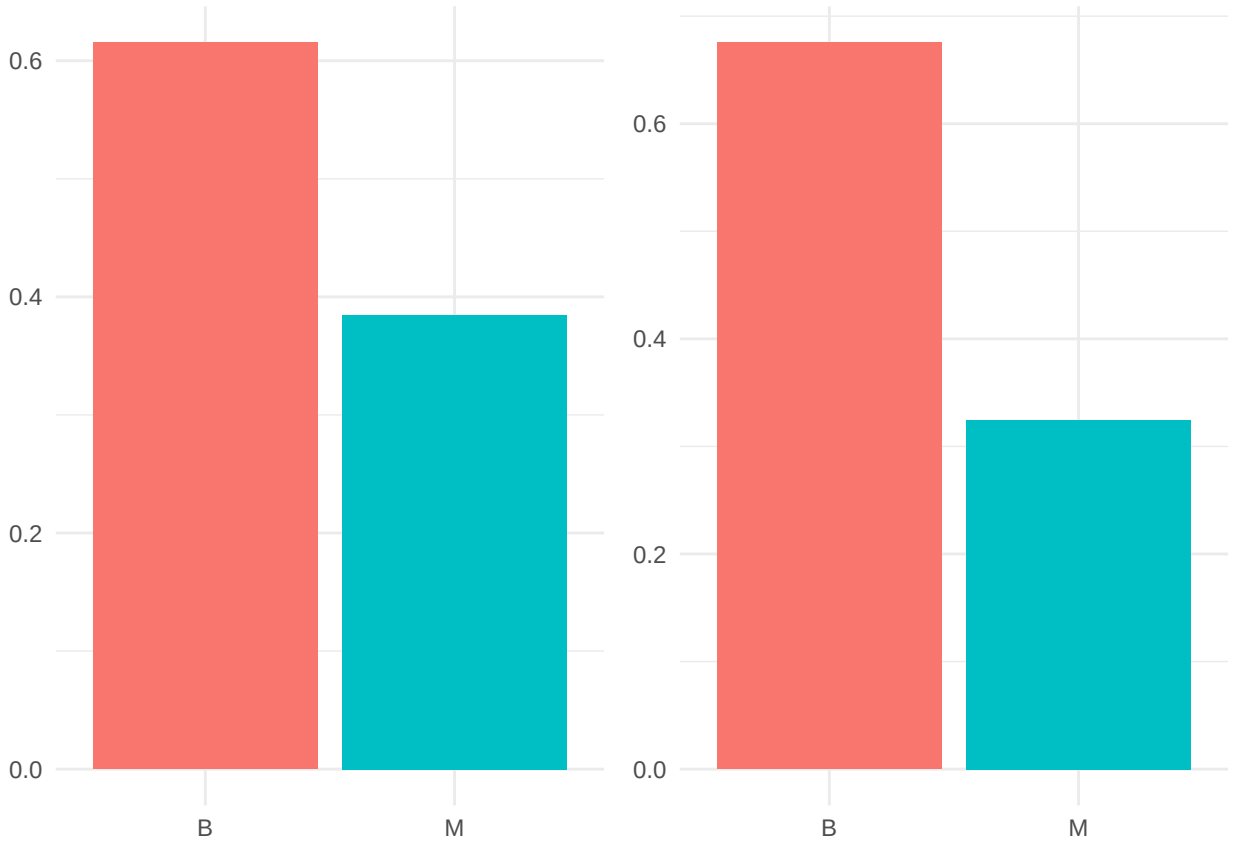
```
sum(is.na(data))
```

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

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

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


texture_se
texture_m
texture_w
area_w
perim_w
radius_w
perim_m
area_m
area_se
radius_se
perim_se
concav_pt_se
compact_m
concav_m
concav_pt_m
concav_pt_w
compact_w
concav_w
compact_se
concav_se
smooth_se
symmetry_se
frac_dim_se
frac_dim_m
frac_dim_w
smooth_m
smooth_w
symmetry_m
symmetry_w
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1


area_m
area_se
area_w
compact_m
compact_se
compact_w
concav_m
concav_pt_m
concav_pt_se
concav_pt_w
concav_se
concav_w
frac_dim_m
frac_dim_se
frac_dim_w
perim_m
perim_se
perim_w
radius_m
radius_se
radius_w
smooth_m
smooth_se
smooth_w
symmetry_m
symmetry_se
symmetry_w
texture_m
texture_se
texture_w
diagnosis B M




```
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)
}
```

Dimensionality Reduction and Feature Selection

PCA

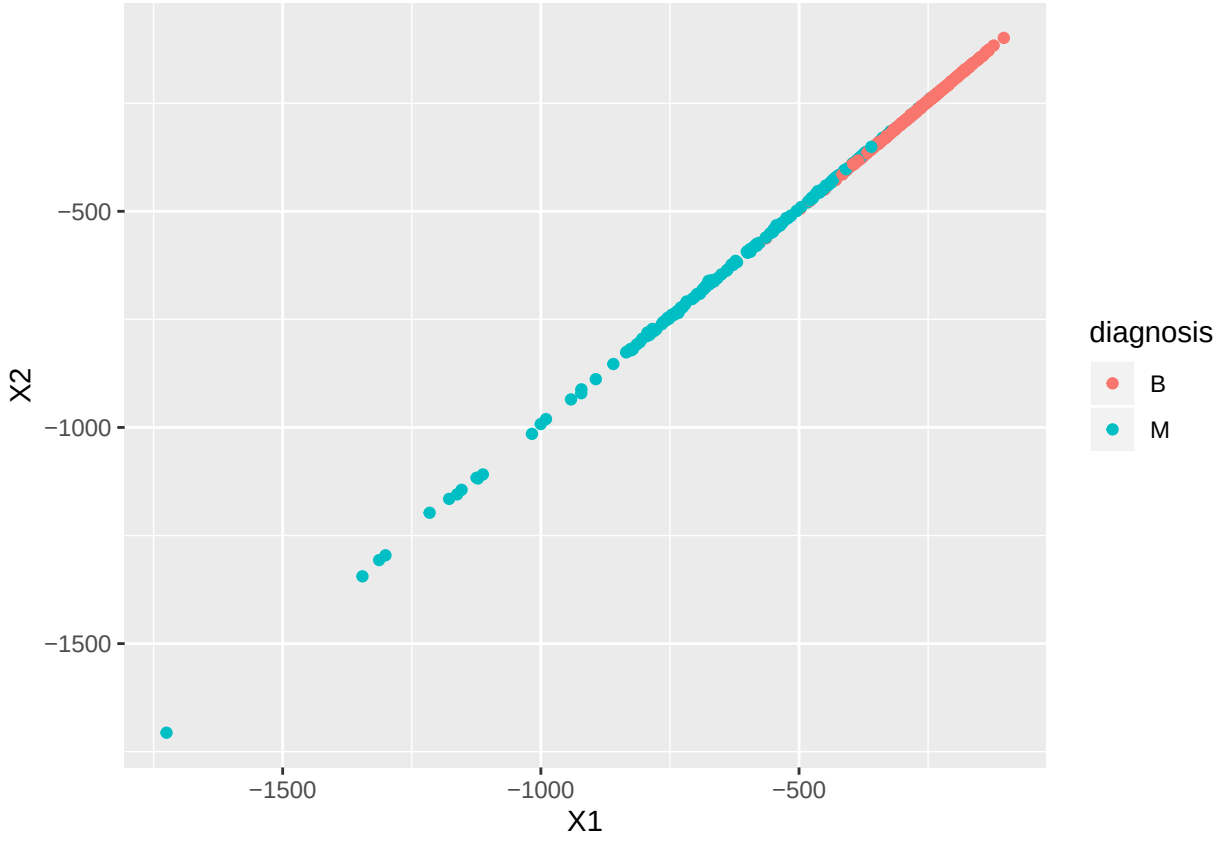
Code

```
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))
```



tSNE

TODO: try different perplexity parameters

```
{r} #reduced_training_data <- tsne::tsne(training_data) # #reduced_train
<- data.frame(cbind(reduced_training_data, training_classes))
#ggplot(data=reduced_training_data, aes(x=X1, y=X2)) + geom_point(aes(co
#
```

- Correlation Feature Selection
- LDA

Classification

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 ϵ_i for each data point into the objective functions and the constraints leading to the optimisation problem:

$$\min_{w,\epsilon_i} \frac{1}{2} w^T w + C \sum_{i=0}^{n} \epsilon_i$$
$$\text{such that } w \cdot x_i + b + \epsilon_i > 1 \text{ if } y_i = 1$$
$$\text{and } w \cdot x_i + b + \epsilon_i < -1 \text{ if } y_i = -1$$

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:

$$\min_{\lambda} \frac{\overline{\lambda} X X^T \overline{\lambda^T}}{4} + \lambda^T \mathbf{1}$$
$$\text{such that } 0 \leq \lambda_i \leq C$$
$$\text{and } \sum_{i}^{n} \lambda_i y_i = 0$$

where

$$X = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \text{ and } \overline{\lambda} = [\lambda_1 \cdot y_i, ..., \lambda_n \cdot y_n] \text{ and } \mathbf{1} = [1, ..., 1] \in \mathbb{R}^{\ltimes}$$

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

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

A and b_0 require slightly more manipulation this time around with

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

and

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

where

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

and

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

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 λ we use the relationship

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

and

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

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

SVM

```
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)
```

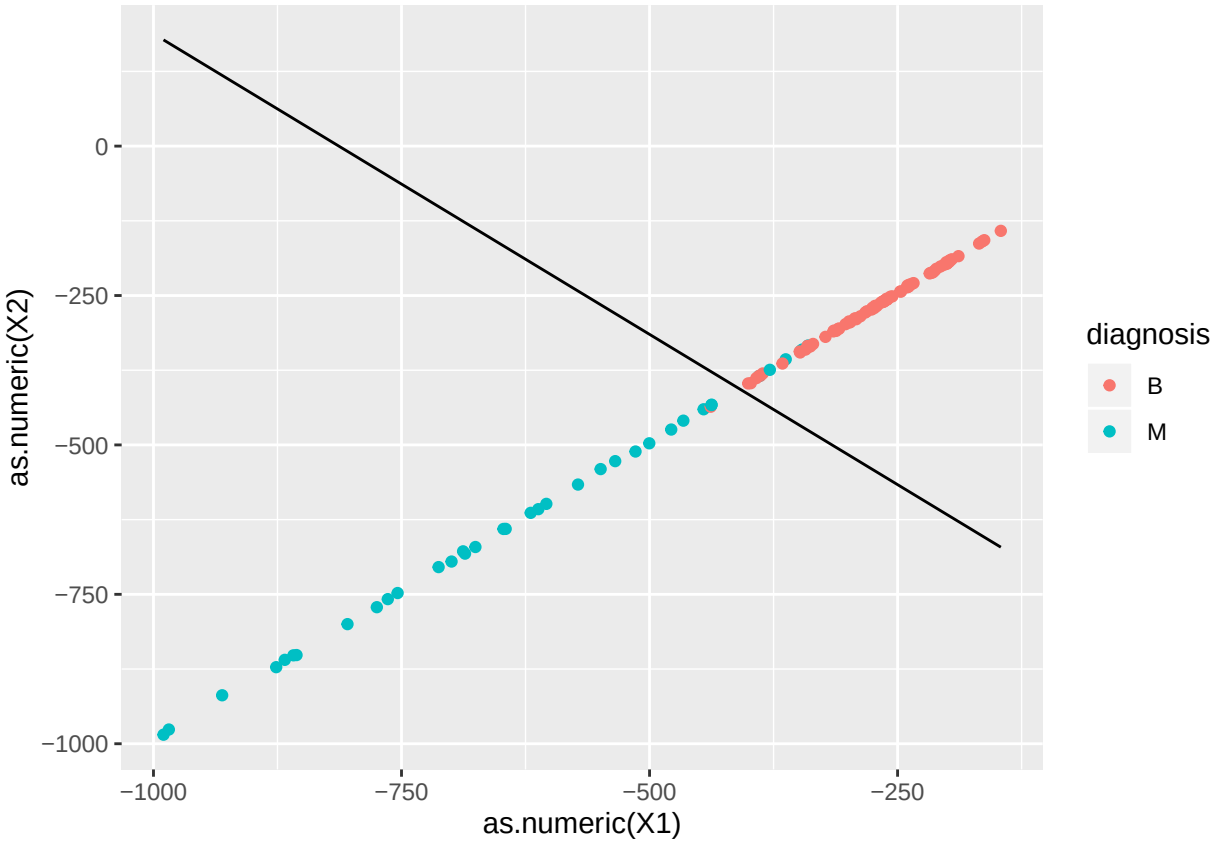
Use PCA then do SVM

```
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))
}
```

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

```
## [1] 93.85965
```

```
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_class

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 $\boldsymbol{x} = (x_1, \cdots x_d)$, where $x_1, \cdots x_d$ are the features. The probability of an observation having label y is given by Bayes rule,

$$\begin{aligned} P(y|x_1, \cdots, x_d) &= \frac{P(x_1, \cdots, x_d|y_k)P(y)}{P(x_1, \cdots, x_d)} \\ &\propto P(x_1, \cdots, x_d|y_k)P(y). \end{aligned}$$

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 *naive* is that very often this assumption is not actually realistic.

This assumption simplifies the posterior to

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

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

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

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 clasess 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)
```

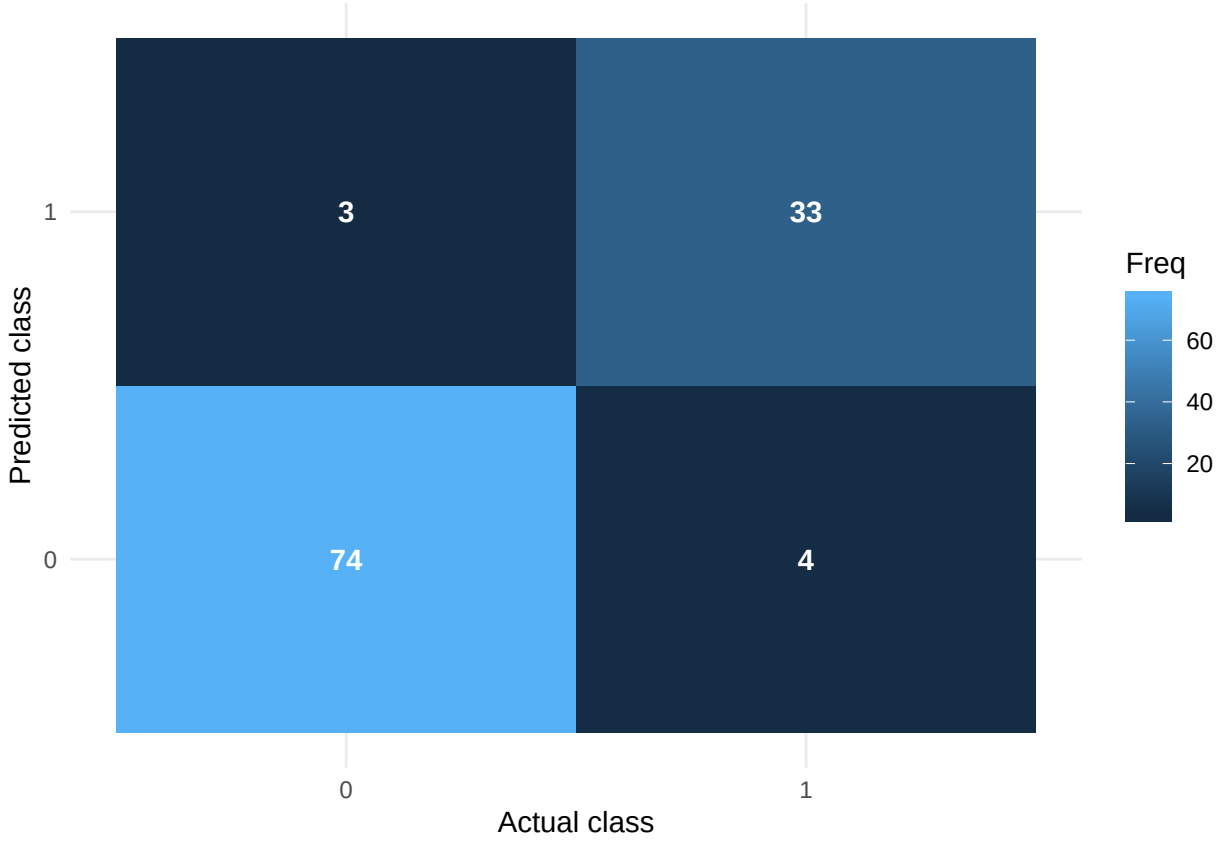
Fit model to dataset

```
install_github("andreabecsek/NaiveBayes")
library(NaiveBayes)
```

```
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)

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

confusion_plot(test_classes,predictions_naive)
```

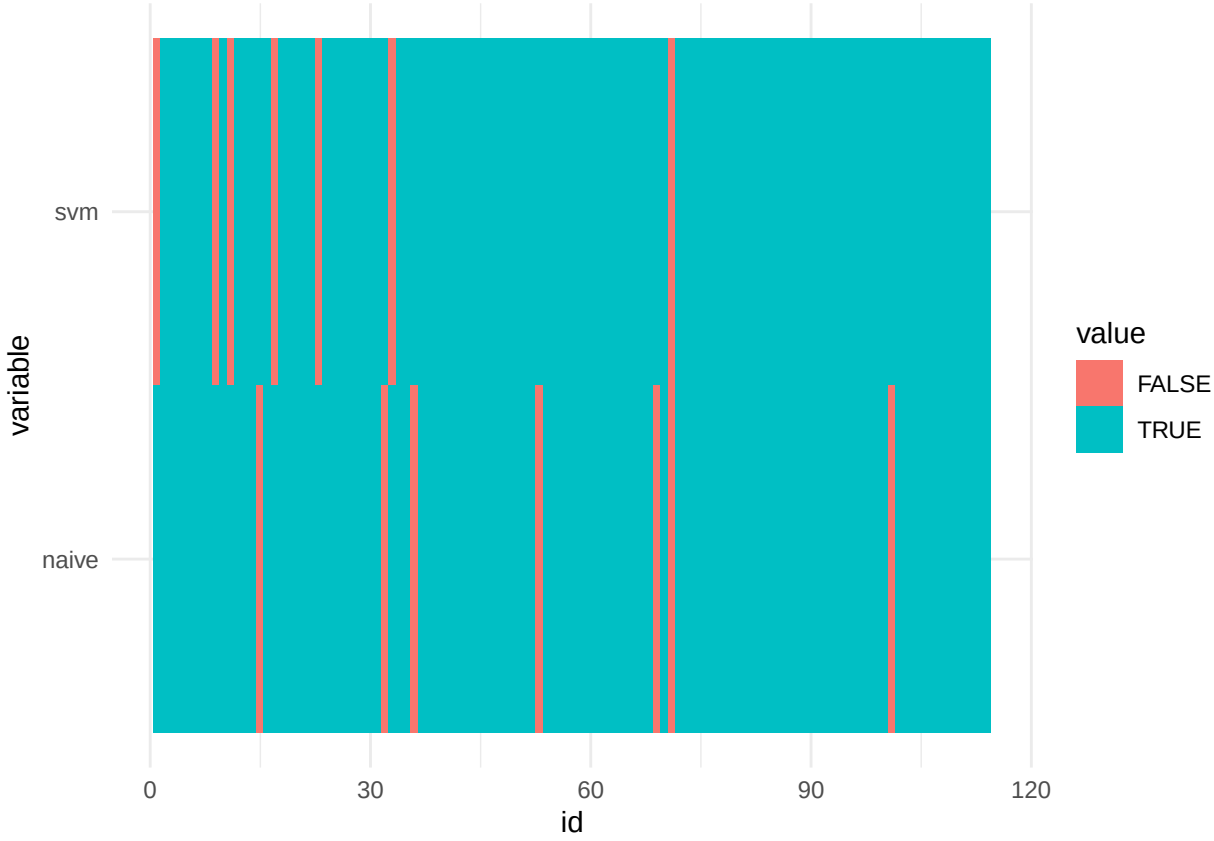


Merge all predictions

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

```
errors <- all_predictions %>%
  mutate(naive=naive==actual) %>%
  mutate(svm=svm==actual) %>%
  dplyr::select(-actual)
```

```
a <- errors %>%
  melt(id='id')

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



TODO: analyse results

- Naive Bayes
- Logistic Regression

Conclusion

- Evaluation of results
- Discuss outliers

TODO: create outlier plot