Week 9 Support Vector Machine (SVM)

Theory and Practice

Ying Lin, Ph.D

March 13, 2020

Overview

- Linearly separable data set and margin maximization
- Non-linearly separable data set soft margin classifier
- Soft margin classifier vs. hard margin classifier
- Kernel trick
- R/Python demo

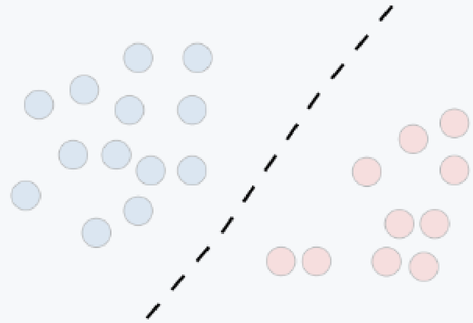
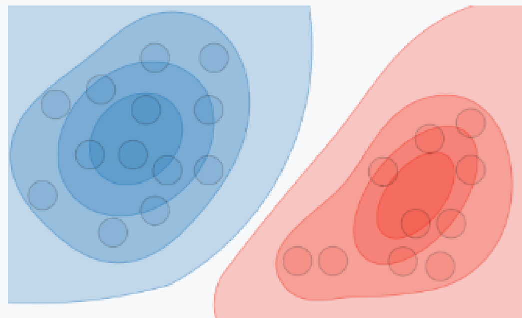
Support Vector Machine (SVM) Motivations

- Work with both linearly separable and linearly non-separable cases
- Maximize margin between categories
- Margins are only decided by the closest data points (support vectors)
    - SVM is robust for the outliers
    - Distance between data points and decision boundary indicates confidence of prediction

Generative vs. Discriminative Algorithms

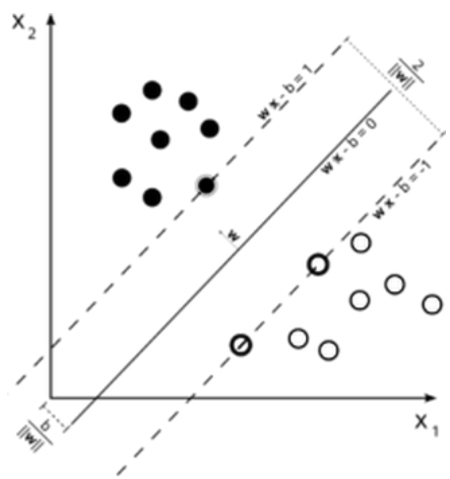
- Generative algorithm: infer the distribution that generate the observed data (joint probability)
  - Model probability of individual classes
  - Example: naive Bayes classifier,
- Discriminative algorithm: identify the decision boundary that separates different classes of the observed data. Make fewer assumptions about the data distribution than generative algorithm
  - Model the (hard or soft) decision boundary between different classes
  - Example: Support Vector Machine, logistic regression, decision tree induction
- Discriminative algorithm usually outperforms generative algorithm, given large enough training dataset

Generative vs. Discriminative Algorithms (2)

	Discriminative model	Generative model
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$ to then deduce $P(y\|x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes

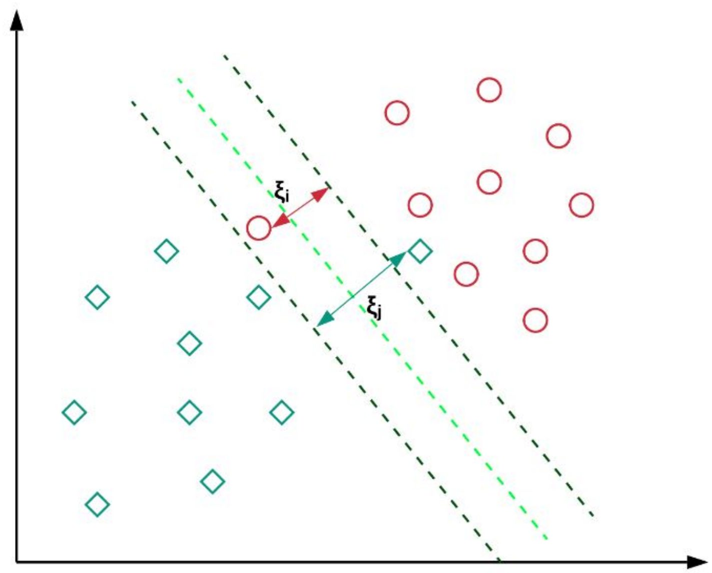
SVM Hyperplane

- Linearly separable cases
- Maximize the margin between two parallel hyperplanes or minimize $||\vec{w}||$
- Support vectors: data points on those two parallel hyperplanes
  - Number of support vectors: complexity of models
- Prediction: hyperplane divides the feature space in half and we can classify new points by determining which side of the hyperplane they fall on



Soft Margin Classifier

- Allow SVM to make a certain number of mistakes (ξ)
- Keep margin as wide as possible so that rest of points can still be classified correctly
- $y_i(\vec{w} \cdot \overrightarrow{x_i} + b) \geq 1 - \xi_i$

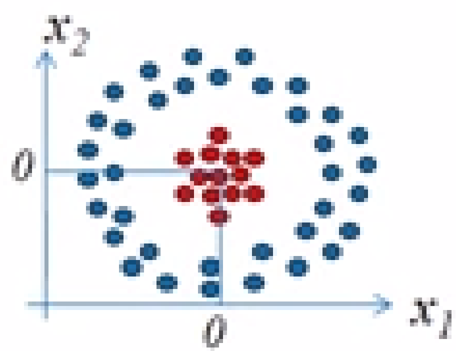


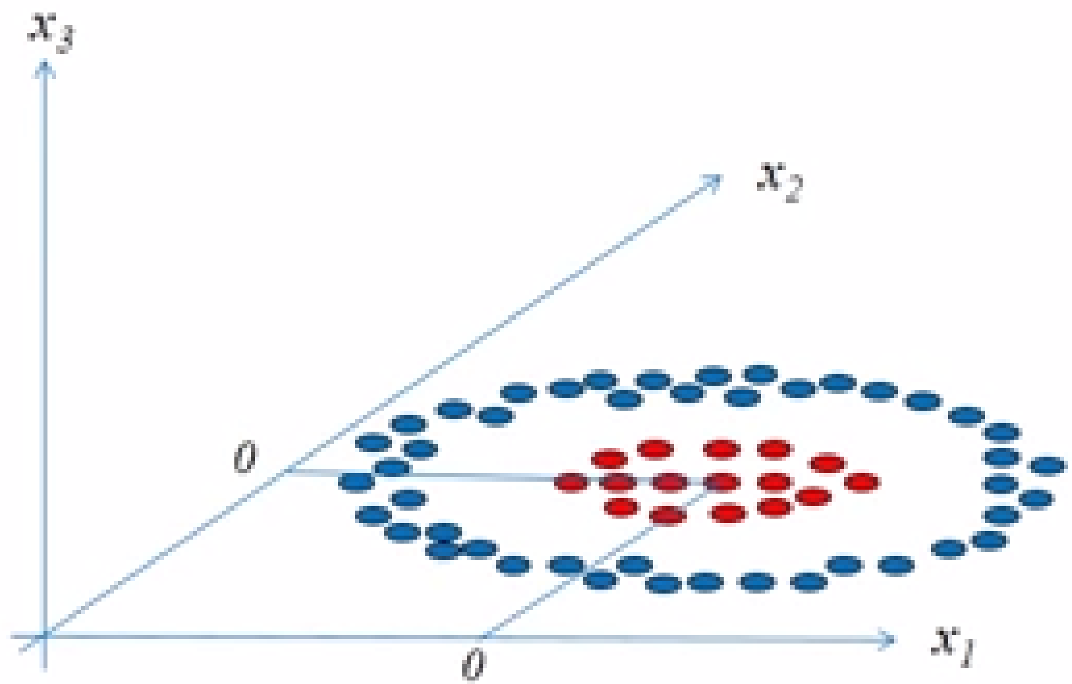
7/35

Loss Function for SVM: Hinge Loss

- SVM loss leads to SVM model with the max-margin property
- Hinge/SVM loss
  - $Hinge(y_i, \hat{y_i}) = \sum_i max(0, 1 - y_i * \hat{y_i})$
  - Scenario 1: y_i = -1, $\hat{y_i}$ = 0.5, Hinge loss is 1.5
  - Scenario 2: y_i = -1, $\hat{y_i}$ = -0.2, Hinge loss is 0.8
  - Scerario 3: y_i = -1, $\hat{y_i}$ = -1.6, Hinge loss is 0
  - Therefore, Hinge loss penalizes those confident misclassification the most, and no penalty on the correct prediction

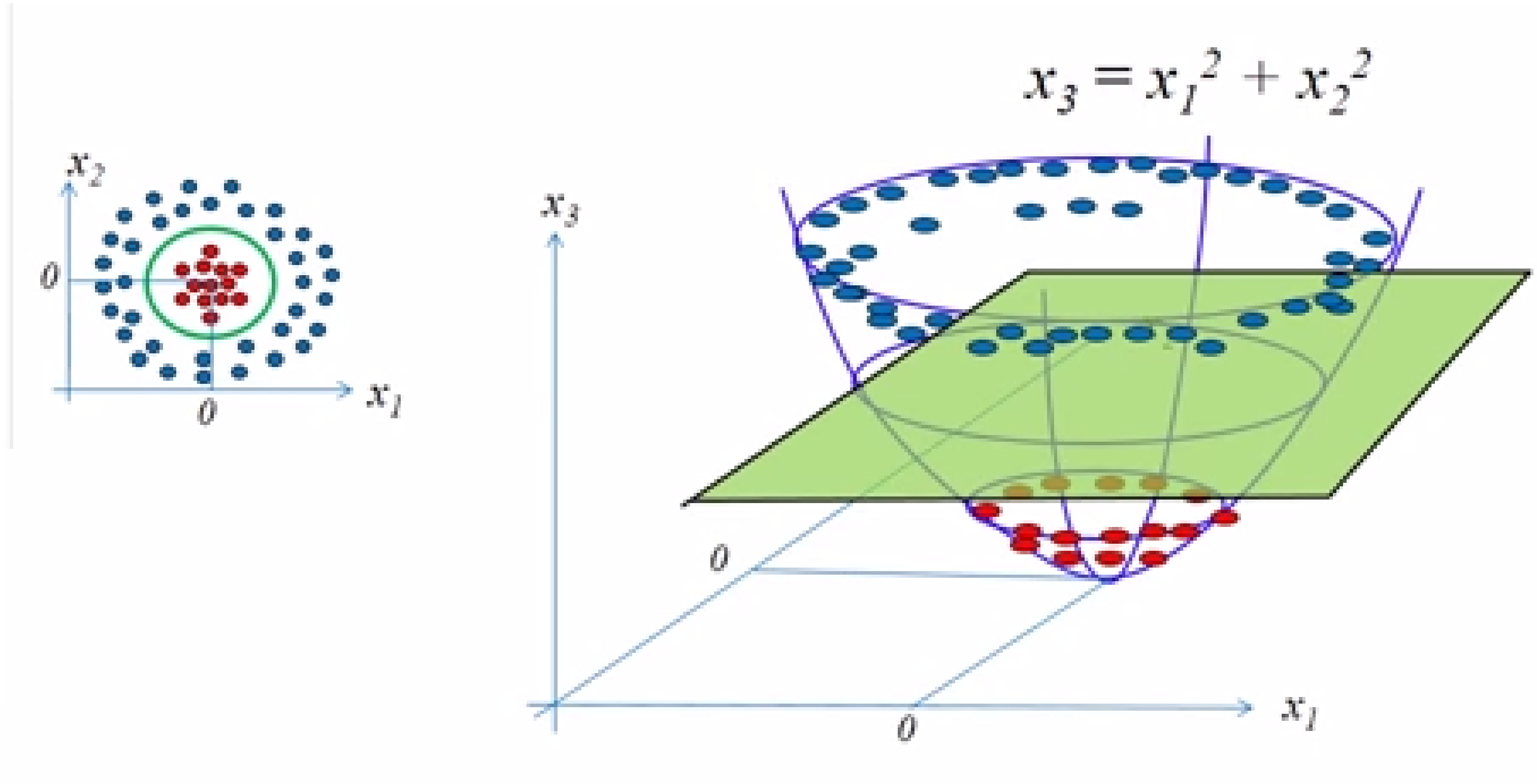
Linearly Inseparable Challenge





9/35

Linearly Inseparable Challenge in a Higher Dimension



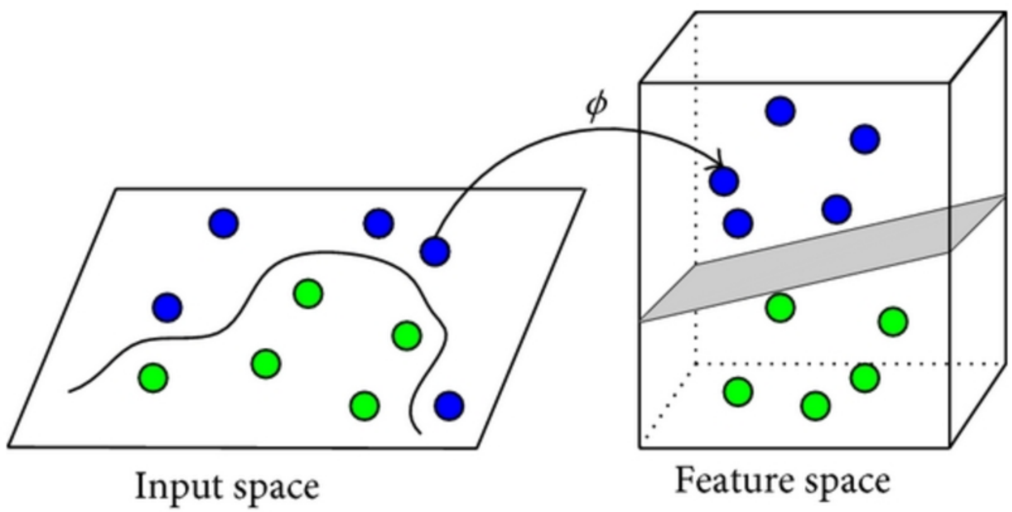
10/35

Kernel Tricks

- Deal with classification that requires multiple (and nonlinear) boundaries
- Ideas: apply mathematical functions (kernels) to project linearly inseparable data points into higher dimensional space so that a hyperplane could be found to separate different classes
  - $K(X_i, X_j) = \phi(X_i) \cdot \phi(X_j)$
  - Why kernel function is a measure of similarity: the inner product above means the projection of $\phi(X_i)$ on $\phi(X_j)$ or how much overlap (or how similar) between $\phi(X_i)$ and $\phi(X_j)$
  - Kernel function is a distance function on a higher dimension where hyperplane could be found.
  - Project the data to the higher dimension space in order to find hyperplane to separate classes without explicitly projecting with ϕ function and calculate the distance in the projected higher dimesnion space

Kernel Trick (2)

- The kernel trick avoids the explicit mapping that is needed to get linear learning algorithms to learn a nonlinear function or decision boundary.
- For all and in the input space , certain functions can be expressed as an inner product in another space . The function is often referred to as a kernel or a kernel function.



Non-linear Kernel Functions

- SVM is about finding a hyperplane (through margin maximization) to separate different classes
- Kernel: find distance between data points on a higher dimensional space where a hyperplane could be identified
  - Radial basis function (RBF): project to infinite dimensional space

$$K(X_m, X_n) = exp(-\frac{||X_m - X_n||^2}{2\sigma^2})$$

  - Polynomial kernel

$$K(X_m, X_n) = (X_m \cdot X_n + c)^d$$

  - Sigmoid kernel

$$K(X_m, X_n) = tanh(\kappa X_m \cdot X_n - \delta)$$

  - Commpare above distance functions with Manhattan and Euclidean distance functions in the current feature space prior to projecting to the higher dimensional space

13/35

Data Preprocessing and Preparation

- Kernel functions are based on distance (distance on higher dimensions) and herefore require the following two preprocessing techniques
- Normalization
- Conversion from categorical attributes to numerical attributes
- Missing value imputation

Model Parameters

- C (Cost): used for regularization, penalty associated with misclassification
  - Soft margin classification: allow a small numbr of misclassifications
  - Hyperparameter controling penalizing use of slack variables ξ
  - Higher C: lower bias but risk of overfitting; higher complexity of decision boundary; higher number of support vectors; Lower C: higher bias but lower variance
- Kernel function specific parameters
  - γ for RBF: contols the tradeoff between error due to bias and variance
    - $k(X_m, X_n) = exp(-\gamma||X_m - X_n||^2)$
    - Higher γ means higher risk of overfitting; lower γ means higher bias; moves in the same direction as C
  - Degree for Polynomial kernel
  - Scale for Polynomial kernel

Properties of SVM

- Pros
  - Excellent performance for a wide variety of tasks because it can deal with nonlinear decision boundary
  - Provide confidence estimate (probability transformed from distance by proper mathematical functions)
  - Robust to noisy data
  - Flexible with data representation: numerical and categorical attributes
  - Effective with low sample/feature ratio data
- Cons
  - Lack of interpretability. Black-box model
  - Risk of choosing wrong kernel function

Demo Dataset: Diabetes

```
# install.packages("mlbench")
library(caret)
library(mlbench)
data("PimaIndiansDiabetes")
diabetes <- PimaIndiansDiabetes
str(diabetes)
```

```
## 'data.frame':    768 obs. of  9 variables:
##  $ pregnant: num  6 1 8 1 0 5 3 10 2 8 ...
##  $ glucose : num  148 85 183 89 137 116 78 115 197 125 ...
##  $ pressure: num  72 66 64 66 40 74 50 0 70 96 ...
##  $ triceps : num  35 29 0 23 35 0 32 0 45 0 ...
##  $ insulin : num  0 0 0 94 168 0 88 0 543 0 ...
##  $ mass    : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
##  $ pedigree: num  0.627 0.351 0.672 0.167 2.288 ...
##  $ age     : num  50 31 32 21 33 30 26 29 53 54 ...
##  $ diabetes: Factor w/ 2 levels "neg","pos": 2 1 2 1 2 1 2 1 2 2 ...
```

```
summary(diabetes)
```

```
##     pregnant         glucose         pressure         triceps
##  Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
```

Train and Test Data Partition

```
library(caret)

set.seed(188)
train_index <- createDataPartition(diabetes$diabetes, p = 0.7, list = FALSE)

diabetes_train <- diabetes[train_index, ]
diabetes_test <- diabetes[-train_index, ]
```

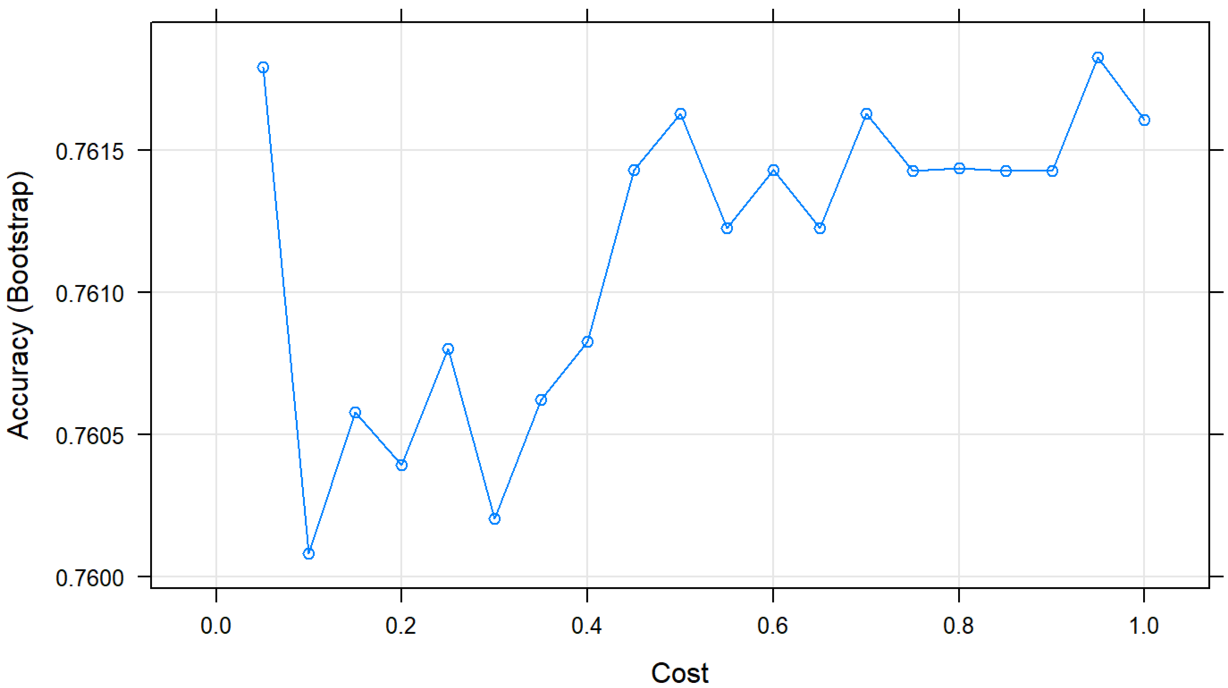
Demo: Train a Linear SVM Model

```
set.seed(1818)
model_svm_linear <- train(diabetes ~ ., data = diabetes_train,
                          method = "svmLinear",
                          preProcess = c("center", "scale"),
                          trControl = trainControl(method = "boot", number = 25),
                          tuneGrid = expand.grid(C = seq(0, 1, 0.05)))
model_svm_linear
```

```
## Support Vector Machines with Linear Kernel
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 538, 538, 538, 538, 538, 538, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.00        NaN        NaN
##   0.05  0.7617913  0.4563736
##   0.10  0.7600813  0.4535191
```

Sensitivity Test for C

```
plot(model_svm_linear)
```



Linear SVM Performance Evaluation: Hold-out Method

```
predict_svm_linear <- predict(model_svm_linear, newdata = diabetes_test)
confusionMatrix(predict_svm_linear, diabetes_test$diabetes)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction neg pos
##        neg 134  36
##        pos  16  44
##
##                Accuracy : 0.7739
##                  95% CI : (0.7143, 0.8263)
##     No Information Rate : 0.6522
##     P-Value [Acc > NIR] : 4.208e-05
##
##                   Kappa : 0.4708
##  Mcnemar's Test P-Value : 0.008418
##
##             Sensitivity : 0.8933
##             Specificity : 0.5500
##          Pos Pred Value : 0.7882
```

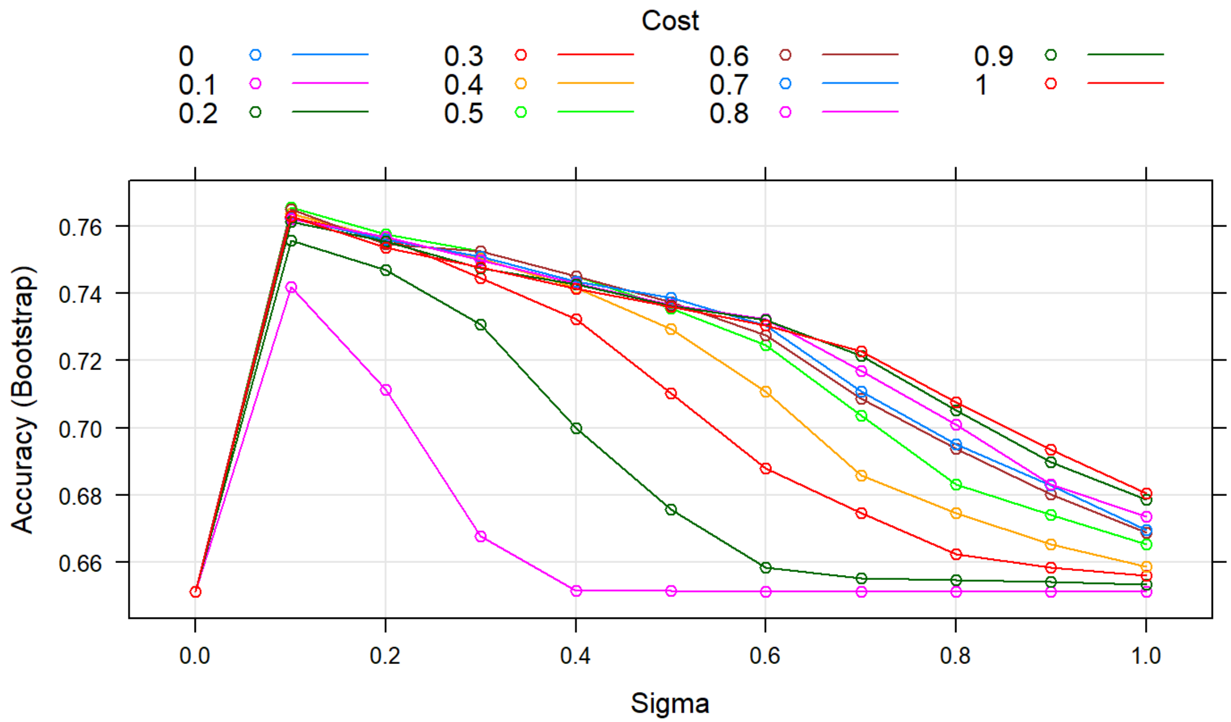
SVM with Non-linear Kernel: RBF

```
model_svm_rbf <- train(diabetes ~ ., data = diabetes_train,
                       preProcess = c("center", "scale"),
                       tuneGrid = expand.grid(sigma = seq(0, 1, 0.1),
                                              C = seq(0, 1, 0.1)),
                       method = "svmRadial",
                       trControl = trainControl(method = "boot",
                                                number = 25))
```

```
model_svm_rbf
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 538, 538, 538, 538, 538, 538, ...
## Resampling results across tuning parameters:
##
##   sigma  C    Accuracy   Kappa
##   0.0    0.0        NaN          NaN
##   0.0    0.1  0.6512478  0.000000000
##   0.0    0.2  0.6512478  0.000000000
##   0.0    0.3  0.6512478  0.000000000
##   0.0    0.4  0.6512478  0.000000000
##   0.0    0.5  0.6512478  0.000000000
##   0.0    0.6  0.6512478  0.000000000
##   0.0    0.7  0.6512478  0.000000000
##   0.0    0.8  0.6512478  0.000000000
##   0.0    0.9  0.6512478  0.000000000
##   0.0    1.0  0.6512478  0.000000000
```

```
plot(model_svm_rbf)
```



```
predict_svm_rbf <- predict(model_svm_rbf, newdata = diabetes_test)
confusionMatrix(predict_svm_rbf, diabetes_test$diabetes)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction neg pos
##        neg 131  43
##        pos  19  37
##
##                Accuracy : 0.7304
##                  95% CI : (0.6682, 0.7866)
##     No Information Rate : 0.6522
##     P-Value [Acc > NIR] : 0.006878
##
##                   Kappa : 0.3611
##  Mcnemar's Test P-Value : 0.003489
##
##             Sensitivity : 0.8733
##             Specificity : 0.4625
##          Pos Pred Value : 0.7529
##          Neg Pred Value : 0.6607
##              Prevalence : 0.6522
##          Detection Rate : 0.5696
##    Detection Prevalence : 0.7565
```

Compare the Performance of Multiple Algorithms

```
model_comparison <- resamples(list(SVMLinear = model_svm_linear,
                                   SVMRBF = model_svm_rbf))
summary(model_comparison)
```

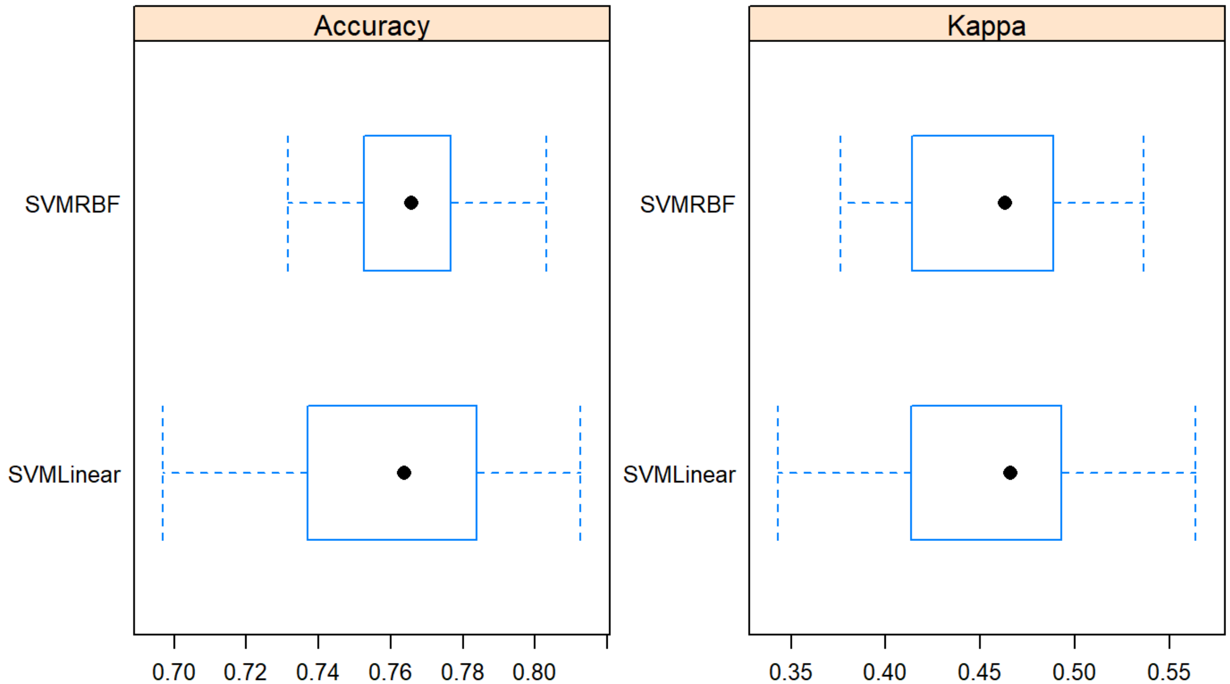
```
##
## Call:
## summary.resamples(object = model_comparison)
##
## Models: SVMLinear, SVMRBF
## Number of resamples: 25
##
## Accuracy
##                Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## SVMLinear 0.6969697 0.7371134 0.7638191 0.7618265 0.7839196 0.8125000    0
## SVMRBF    0.7315789 0.7526316 0.7656250 0.7654968 0.7766497 0.8031088    0
##
## Kappa
##                Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## SVMLinear 0.3432836 0.4134789 0.4659401 0.4582616 0.4931610 0.5637772    0
## SVMRBF    0.3760464 0.4141189 0.4633530 0.4528180 0.4885286 0.5365331    0
```

26/35

Graphically Compare Performance

```
scales <- list(x = list(relation = "free"),
               y = list(relation = "free"))

bwplot(model_comparison, scales = scales)
```



Load in Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn import svm
from sklearn import model_selection
from sklearn.model_selection import GridSearchCV
from matplotlib.legend_handler import HandlerLine2D
```

Data Splitting and Preparation

```
diabetes = r.diabetes
# diabetes = pd.read_csv("https://datahub.io/machine-learning/diabetes/r/diabetes.csv")
X = diabetes.drop("class", axis=1)
y = diabetes["class"]
X_train, X_test, y_train,y_test = train_test_split(X, y, test_size=0.3, random_state=1)
scaler = StandardScaler()
scaler.fit(X_train)
```

```
## StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
X_train_std = scaler.transform(X_train)
X_test_std = scaler.transform(X_test)
```

29/35

Linear SVM Model Tuning

```
lin_clf = svm.LinearSVC()
lin_clf
```

```
## LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
##           intercept_scaling=1, loss='squared_hinge', max_iter=1000,
##           multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
##           verbose=0)
```

```
bs = model_selection.ShuffleSplit(n_splits=25,test_size=0.3,random_state=0)
param_grid = {'C':[0.25,0.5,0.75,1],'penalty':['l2']}

gridbs = GridSearchCV(lin_clf,param_grid,cv=bs)
gridbs.fit(X_train_std,y_train)
```

```
## GridSearchCV(cv=ShuffleSplit(n_splits=25, random_state=0, test_size=0.3, train_size=None),
##              error_score='raise-deprecating',
##              estimator=LinearSVC(C=1.0, class_weight=None, dual=True,
##                                  fit_intercept=True, intercept_scaling=1,
##                                  loss='squared_hinge', max_iter=1000,
##                                  multi_class='ovr', penalty='l2',
##                                  random_state=None, tol=0.0001, verbose=0),
##              iid='warn', n_jobs=None,
```

30/35

Hyper-Parmater C Sensitivity

```
C = np.arange(0.05,1,0.05)
train_results_svm =[]
test_results_svm = []
for n in C:
   model = svm.LinearSVC(C=n)
   model.fit(X_train_std, y_train)
   acc = cross_val_score(model, X_train_std, y_train, cv=bs).mean()*100
   train_results_svm.append(acc)
```

```
## LinearSVC(C=0.05, class_weight=None, dual=True, fit_intercept=True,
##           intercept_scaling=1, loss='squared_hinge', max_iter=1000,
##           multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
##           verbose=0)
## LinearSVC(C=0.1, class_weight=None, dual=True, fit_intercept=True,
##           intercept_scaling=1, loss='squared_hinge', max_iter=1000,
##           multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
##           verbose=0)
## LinearSVC(C=0.15000000000000002, class_weight=None, dual=True,
##           fit_intercept=True, intercept_scaling=1, loss='squared_hinge',
##           max_iter=1000, multi_class='ovr', penalty='l2', random_state=None,
##           tol=0.0001, verbose=0)
## LinearSVC(C=0.2, class_weight=None, dual=True, fit_intercept=True,
##           intercept_scaling=1, loss='squared_hinge', max_iter=1000,
```

Linear SVM Performance Evaluation

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
y_pred = model.predict(X_test_std)
cmnb=confusion_matrix(y_test, y_pred, labels=["tested_negative","tested_positive"])
target_names = ["tested_negative","tested_positive"]
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
##                  precision    recall  f1-score   support
##
## tested_negative       0.79      0.90      0.84       146
## tested_positive       0.78      0.58      0.66        85
##
##        accuracy                           0.78       231
##       macro avg       0.78      0.74      0.75       231
##    weighted avg       0.78      0.78      0.78       231
```

32/35

SVM with RBF Kernel Function: γ

```
rbf_clf = svm.SVC()
gamma = [0.001,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
train_results_svm =[]
train_results_svmC2 = []
train_results_svmC3= []
for n in gamma:
   model1 = SVC(C=0.1,kernel='rbf',gamma=n,cache_size=2000)
   model1.fit(X_train,y_train)
   acc = cross_val_score(model1, X_train_std, y_train, cv=bs).mean()*100
   train_results_svm.append(acc)
   model2 = SVC(C=0.5,kernel='rbf',gamma=n,cache_size=2000)
   acc2 = cross_val_score(model2, X_train_std, y_train, cv=bs).mean()*100
   train_results_svmC2.append(acc2)
   model3 = SVC(C=1,kernel='rbf',gamma=n,cache_size=2000)
   acc3 = cross_val_score(model3, X_train_std, y_train, cv=bs).mean()*100
   train_results_svmC3.append(acc3)
```

```
## SVC(C=0.1, cache_size=2000, class_weight=None, coef0=0.0,
##     decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
##     max_iter=-1, probability=False, random_state=None, shrinking=True,
##     tol=0.001, verbose=False)
## SVC(C=0.1, cache_size=2000, class_weight=None, coef0=0.0,
##     decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
```

RBM Kernel SVM Performance

```
param_grid = {'C':[0.5], 'gamma':[0.1]}
gridrbf = GridSearchCV(model2, param_grid, cv=bs)
gridrbf.fit(X_train_std, y_train)
```

```
## GridSearchCV(cv=ShuffleSplit(n_splits=25, random_state=0, test_size=0.3, train_size=None),
##              error_score='raise-deprecating',
##              estimator=SVC(C=0.5, cache_size=2000, class_weight=None, coef0=0.0,
##                            decision_function_shape='ovr', degree=3, gamma=1,
##                            kernel='rbf', max_iter=-1, probability=False,
##                            random_state=None, shrinking=True, tol=0.001,
##                            verbose=False),
##              iid='warn', n_jobs=None, param_grid={'C': [0.5], 'gamma': [0.1]},
##              pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
##              scoring=None, verbose=0)
```

34/35

RBM Kernel SVM Performance

```
y_pred2 = gridrbf.predict(X_test_std)
cmnb2=confusion_matrix(y_test, y_pred2, labels=["tested_negative", "tested_positive"])
target_names = ["tested_negative", "tested_positive"]
print(classification_report(y_test, y_pred2, target_names=target_names))
```

```
##                  precision    recall  f1-score   support
##
## tested_negative       0.78      0.92      0.85       146
## tested_positive       0.81      0.55      0.66        85
##
##        accuracy                           0.79       231
##       macro avg       0.80      0.74      0.75       231
##    weighted avg       0.79      0.79      0.78       231
```