Math 533 Midterm Project: Support Vector Machine

Leticia Han

10/15/2020

Contents

1 Overview/Introduction

1.1 History

In 1936, Ronald A. Fisher introduced the first classification algorithm with a linear discriminant function using his famous `iris` data, which led to the generalization known today as the linear discriminant analysis [10, 6]. It is considered a linear classifier for a binary classification that applies Bayes' theorem. Thus, Cortes and Vapnik [6] assert that classification algorithms began with the idea of solving for linear decision boundaries. Less than 30 years later, in 1962, Frank Rosenblatt created a different algorithm for a binary classification called perceptron, which is a simple feedforward neural network that "implements a piecewise linear separating surface" [6]. When backpropagation algorithm was invented in 1986, it was "considered a breakthrough in the early days of neural networks, since it made fitting a complex model computationally manageable" [9]. The statistics community was amazed at its scalability to large dimensions. Its complex, hierarchical function of the feature space was still a piecewise linear type decision function [9].

With this groundbreaking algorithm, neural networks dominated the field of machine learning from the late 1980s until early 1990s with AT&T Bell Laboratories as was one of the leading groups in the community. During this time, neural networks was being used to tackle an optical character recognition (OCR) problem which involved classifying handwritten zip codes for US Post Office [9]. In 1992, Vladmik Vapnik and his team from the lab wrote a seminal paper on a new algorithm called the optimal margin classifier that also incorporated the kernel method [2]. Expanding on this algorithm, Vapnik and Cortes invented a new machine learning tool known as support vector machine in 1995 that brought a new wave of excitement to the field, by introducing the idea of soft margin. Thus, the algorithm was presented as a combination of the optimal margin classifier, soft margin, and the kernel method. Not only did they show that the algorithm is extremely powerful and universal, it outperformed neural networks in handwritten digit recognition [6]. The success and popularity of the support vector machine is attributed to its good performance, applicability to a variety of settings, and its novel approach to classification. It is now considered one of the best machine learning algorithm.

1.2 Theory

The support vector machine is based upon the idea of a separating hyperplane, and with certain conditions and additional functions, creates a decision rule to classify an observation in a two-class setting. The separating hyperplane is extended to idea of the optimal hyperplane and, along with the notion of soft margin and the utilization of the kernel method into the algorithm, became the support vector machine.

1.2.1 Hyperplane and Maximal Margin Classifier

A hyperplane is a flat subspace of dimension $p - 1$ in a p-dimensional space [14]. In its simplest form, it is a line in a two-dimensional space and can be defined by the equation

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0 \tag{1}$$

where β_i's are parameters and any $x = (x_1, x_2)^T$ satisfying (1) is a point on the hyperplane. More generally, in a p-dimensional space, the hyperplane is defined by

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p = 0 \tag{2}$$

which can be rewritten as

$$\{x : f(x) = x^T \beta + \beta_0 = 0\}, \tag{3}$$

where $x \in \mathbb{R}^p$ and β is a unit vector: $\|\beta\| = 1$.

Any x that does not satisfy (3), then, must be on either side of the hyperplane

$$x^T\beta + \beta_0 < 0, \tag{4}$$

$$x^T\beta + \beta_0 > 0, \tag{5}$$

which means that the hyperplane can be thought of as a linear surface dividing the p-dimensional space into two. Suppose we have training data with n observations and p predictors, $x_i \in \mathbb{R}^p$ for $i = 1, \ldots, n$, and these observations belong to two classes, i.e., $y_1, \cdots, y_n \in \{-1, 1\}$. Furthermore, suppose the training data is linearly separable. Then, we can construct a hyperplane that can perfectly separate data into their two class labels. Then, the separating hyperplane has the property

$$x^T\beta + \beta_0 > 0 \text{ if } y = 1, \tag{6}$$

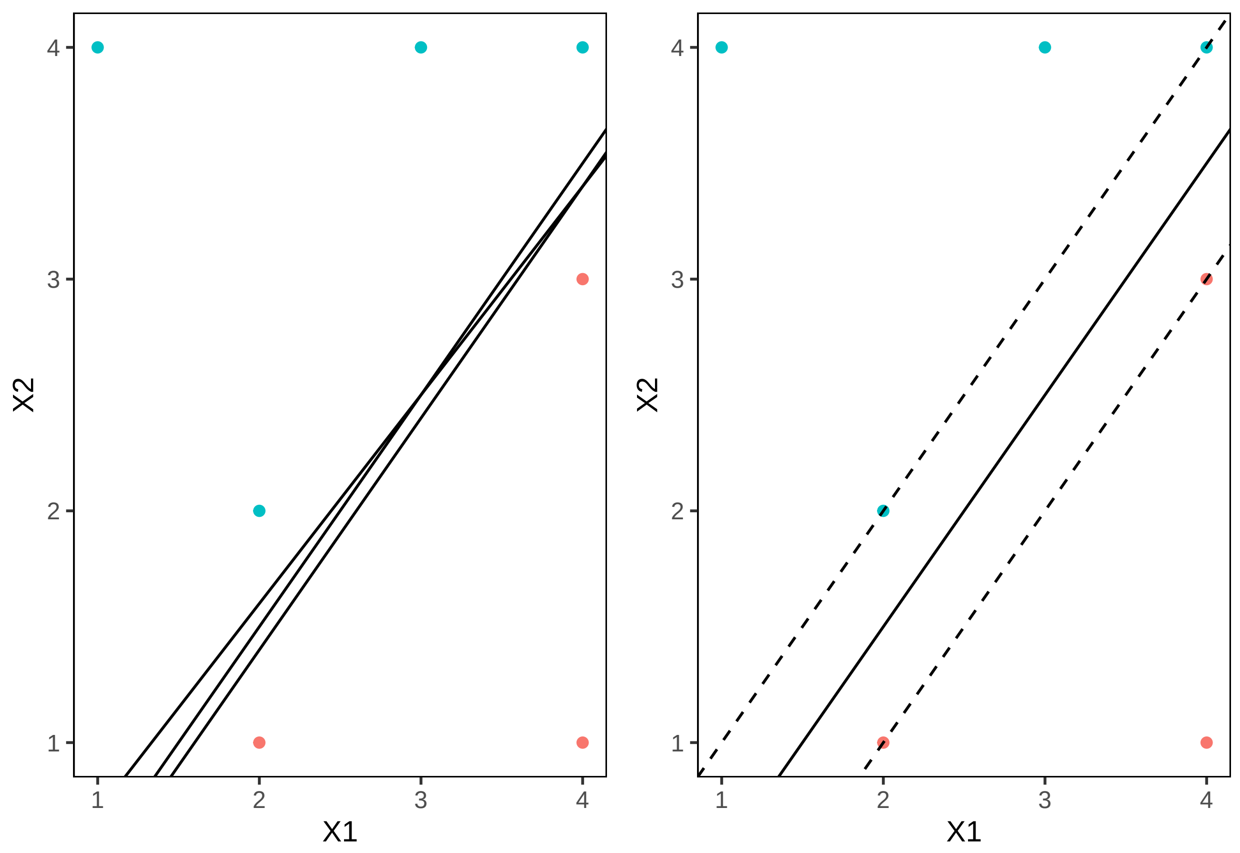
$$x^T\beta + \beta_0 < 0 \text{ if } y = -1, \tag{7}$$

which can be combined and rewritten as

$$y(x^T\beta + \beta_0) > 0 \tag{8}$$

If the data is linearly separable, a separating hyperplane exists, which we can use to construct a linear classifier. Suppose we have a test observation x^* with p features, which we want to classify as either -1 or 1. By solving for (8), x^* is assigned to either class -1 or class 1 depending on which side of the hyperplane it falls onto. Therefore, the decision rule is based on the sign of $f(x) = x^T\beta + \beta_0$. Furthermore, the value of $f(x^*)$ tells us how far x^* is far from the hyperplane and determine the level of confidence we can have on the classification result.

In general, if there is perfect separation in our data, then there can exist an infinite number of separating hyperplanes, such as the figure on the left panel below. The best separating hyperplane happens to be the maximal margin hyperplane, also known as optimal separating hyperplane. If we compute the perpendicular distance from each data point to a given hyperplane, we can find the smallest distance from the data points to the hyperplane, which is called the margin M. The optimal separating hyperplane is the one that maximizes the margin, i.e., the "farthest minimum distance to the training observations" [14]. In the figure below on the right, the separating hyperplane is the solid black line and the margin is the width from the dashed line to the solid line. The total width of the optimal separating hyperplane is $2M$.



After obtaining the optimal separating hyperplane, we can then construct a maximal margin classifier that classifies our test data x^* based on which side of the optimal separating hyperplane it falls. The idea is that if the classifier has a large margin on the training data, then we hope it also has a large margin on the test data allowing for new data to be classified correctly. Furthermore, note that the optimal separating hyperplane is only dependent on a small subset of the whole training data, which are the data points lying on the dashed lines in the figure above. The data points that determine the margin for the optimal hyperplane are called the support vectors . The advantage of this is that the optimal separating hyperplane is only affected by the movements of these support vectors and not the rest of the data points, assuming that the other data points do not cross over their margin's boundary. Therefore, the classifier only depends on the support vectors and not the whole training data, which can scale well to large dimensions [6]. However, since the algorithm to find the support vectors need all of the training data, this can be be computationally expensive for a very large n.

On a side note, classifiers that compute the linear combination of the input features and return the sign were called perceptrons, based on Rosenblatt's perceptrons, which set the foundations for neural networks in the 1980s and 1990s [11, 6]. Rosenblatt's perceptron learning algorithm finds a separating hyperplane by trying to minimize the distance of misclassified points to the decision boundary. However, there are problems that arise with this algorithm [11]. First, we know that if a separating hyperplane exists, there can be an infinite number of solutions. Second, the algorithm can converge to a separating hyperplane very slowly if the gap is small, therefore, can be time consuming. Another problem is in the case of non-separable data, where the algorithm will fail to converge. The solution to the first problem is the added constraints to the separating hyperplane.

To obtain the maximal margin classifier, we need to find the parameters β_0, β, where $\beta = (\beta_1, \ldots, \beta_p)$, from the training data that maximizes the margin M. In other words, the maximal margin hyperplane is the solution to the optimization problem:

$$\underset{\beta_0,\beta,\|\beta\|=1}{\text{maximize}}\ M \tag{9}$$

$$\text{subject to } y_i(x_i^T\beta+\beta_0) \geq M\ \forall\ i=1,\ldots,n \tag{10}$$

The constraint (10) is to ensure that all the observations are on the correct side of the hyperplane defined by β and β_0 with a distance at least the width of margin M away from this decision boundary, provided that M is positive. And we want to find the parameters that maximize this cushion. The constraint $\|\beta\|=1$ is not a constraint on the hyperplane, but adds meaning to the constraint (10). Since the hyperplane is defined by $f(x)=\beta^Tx+\beta_0=0$, for any two points x_1, x_2 on the hyperplane, $\beta^T(x_1-x_2)=0$, thus the unit vector normal to the boundary is $\beta/\|\beta\|$. If x_0 is any point on the boundary, i.e., $f(x)=0$, then $\beta^Tx_0=-\beta_0$. To find the distance from a point x to the boundary, we project $(x-x_0)$ onto the normal, and we get

$$\begin{aligned}\frac{\beta^T(x-x_0)}{\|\beta\|} &= \frac{1}{\|\beta\|}(\beta^Tx+\beta_0)\\ &= \frac{1}{\|\beta\|}f(x),\end{aligned} \tag{11}$$

which is the signed distance of any point x to the hyperplane since $f(x)$ will be either ±1 based on what side of the hyperplane it falls on. Then, we can rewrite the optimization problem as:

$$\underset{\beta_0,\beta}{\text{maximize}}\ M \tag{12}$$

$$\text{subject to } \frac{1}{\|\beta\|}y_i(x_i^T\beta+\beta_0) \geq M\ \forall\ i=1,\ldots,n \tag{13}$$

Using a rescaling argument, this is equivalent to

$$\underset{\beta_0,\beta}{\text{minimize}}\ \|\beta\| \tag{14}$$

$$\text{subject to } y_i(x_i^T\beta+\beta_0) \geq 1\ \forall\ i=1,\ldots,n, \tag{15}$$

which is now a convex optimization problem since it involves quadratic criterion with linear inequality constraints. The solution involves minimizing Lagrange primal function and then maximizing the Lagrange dual problem as well as the Karush-Kuhn-Tucker conditions [11, 9].

The resulting optimal separating hyperplane produces a function $f(x)=\beta_0+x^T\beta$ for which the classifier itself is $C(x)=sign[f(x)]$. In the case of logistic regression, when a separating hyperplane exists, logistic regression and the maximal margin classifier will have similar results, since the log-likelihood can be driven to 0 and logistic regression can always find it [11].

1.2.2 Soft Margin

The maximal margin classifier is constructed so that no training errors are made. In other words, support vectors strictly determine the margin and no other observations are allowed to fall inside the margin, which means perfect separation of our training data. We can call this the hard margin. Hard margins can lead to problems. First, it can lead to overfitting and may not classify well for test data if margin isn't large enough [9, 14]. Second, if the data is not separable, the algorithm will fail to find a separating hyperplane, since it does not exist. In 1995, Cortes and Vapnik introduced the notion of soft margin [6]. If perfect separation is impossible, we can instead construct a "soft margin hyperplane", which separates the two classes with a minimal number of errors. The resulting classifier is known as the support vector classifier or the soft margin classifier [11, 9, 14].

Instead of finding the optimal separating hyperplane that separates training data without any of the training data falling within the margin, the support vector classifier allows for a small amount of the data to violate the margin boundary, and maybe even the hyperplane itself, thus called the soft margin. Training data on the wrong side of the hyperplane are misclassified data points. Although the classifier is imperfect due to the small number of violations by the training data, it has an advantage that it can be more robust to new data, improved classification performance, and increased flexibility to the classifier.

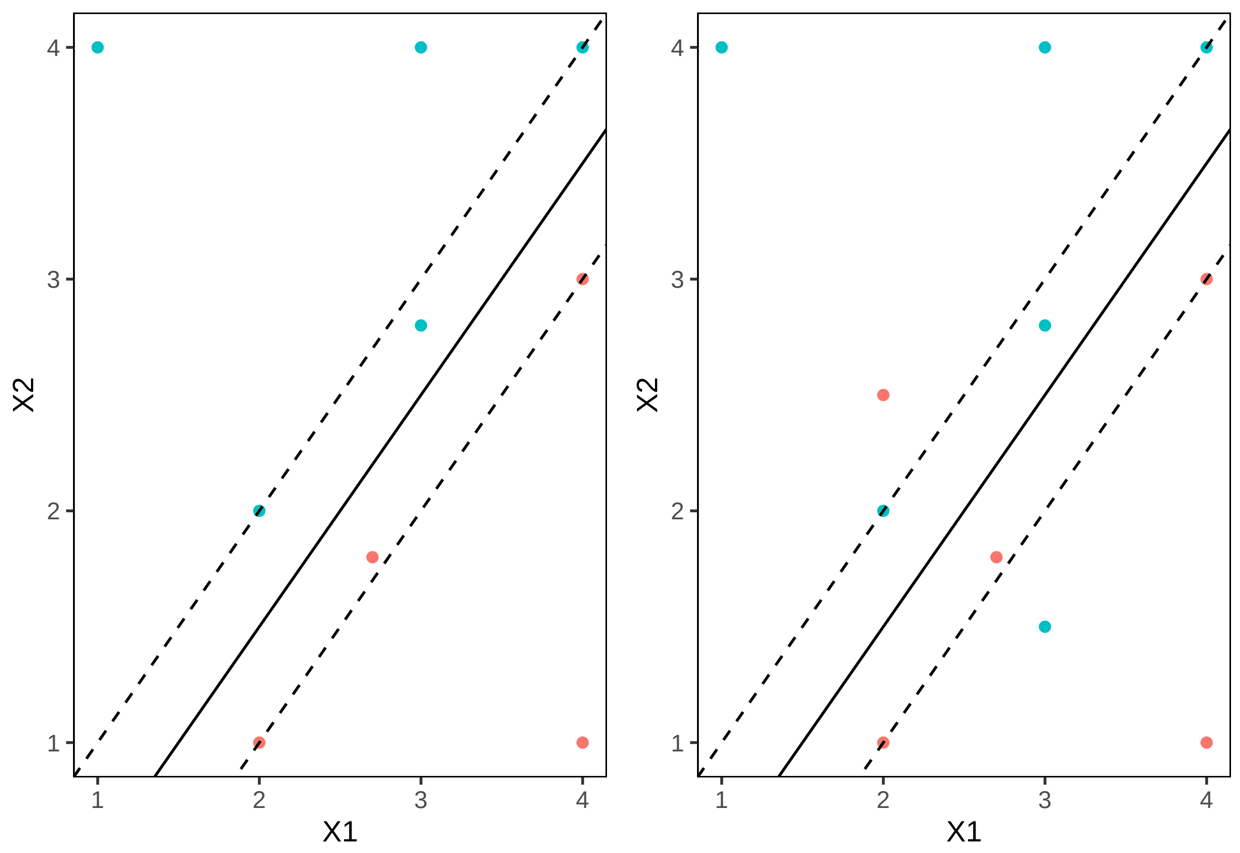
The soft margin hyperplane is the solution to the optimization problem:

$$\underset{\beta_0,\beta,\|\beta\|=1}{\text{maximize}}\ M \tag{16}$$

$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq M(1-\epsilon_i)\ \forall i = 1, \tag{17}$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C, \tag{18}$$

which is similar to the optimal separating hyperplane, but now we have a nonnegative tuning parameter C ("cost") and slack variable ϵ_i. We still want to maximize the margin M, but now we allow for some violations for the training data. The ϵ_i in (17) is the proportional amount by which the classification of x_i is on the wrong side of the margin. It tells us the location of the ith observation relative to the hyperplane and the margin. For an ith observation, if $\epsilon_i = 0$, it is on the correct side of the margin, if $\epsilon > 0$, then it is on the wrong side of the margin but still on the correct side of the hyperplane, and if $\epsilon > 1$, it is on the wrong side of the hyperplane. The left figure below shows two observations violating the margin boundary but not the hyperplane. The right figure shows two additional observations violating the hyperplane. The tuning parameter C is the budget for the total amount of overlap. We are bounding the $\sum \epsilon_i$ by allowing the total number of misclassifications to be at most C. In other words, C determines the tolerable number and severity of margin violations. If $C = 0$, no violations are allowed, which is the same as the optimal margin hyperplane. As we increase C, we are allowing more violations, therefore the width of the margin increases. When C is small, we have narrower margins with less violations, which can lead to low bias but high variance since the classifier is highly fit to the data. When C is large, we have wider margins with more violations, and the classifier can potentially be more biased but may have lower variance.



Similar to the optimal separating hyperplane, we can rewrite the optimization problem with a rescaling argument as:

$$\underset{\beta_0,\beta}{\text{minimize}} \ \|\beta\| \tag{19}$$

$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1 - \epsilon_i \ \forall i \tag{20}$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C, \tag{21}$$

Similar to the optimal separating hyperplane, this is a convex optimization problem (quadratic with linear inequality constraints) that can be solved using Lagrange multipliers [11]. The resulting classifier for test data is $G(x) = sign[f(x)]$, with cost parameter C is the tuning parameter.

The optimization problem has a very attractive property. Any points that are strictly located on the correct side of the margin do not play a role in determining the boundary. Only observations that lie on the margin or that violate the margin affects the hyperplane. As long as it is still on the correct side of the hyperplane and the margin, any movements to the correctly classified data will not affect the classifier at all. For the support vector classifier, the support vectors are the observations that lie on the margin and violate the margin. Thus, only the support vectors play a role in the classifier. If C is small, margin is narrower, thus we have fewer support vectors. If C is large, margin is wider, thus we have more support vectors. Since decision rule is dependent upon the support vectors only, we only need a subset of the training observations. This allows for the support vector classifier to be robust to the movements of observations far away from the decision boundary.

1.2.3 Kernel Method

So far we've been working in the linear case. However, classes can overlap where the boundary between the two classes is nonlinear. In the case of nonlinear decision boundaries, we can enlarge the feature space using basis expansions. Then, in this new feature space, we can find a hyperplane that separates the two class, which we can translate back onto the original feature space, and thus gives us a non-linear decision boundary. This method is called the kernel trick. This is analogous to using polynomials in linear regression to address non-linearity. The kernel trick allows us to work with large dimensions, even infinite dimensions in some cases. Once the basis functions $h_m(x), m = 1, \ldots, M$ are selected, we fit the support vector classifier using the transformed input features $h(x_i) = (h_1(x_i), \ldots, h_M(x_i)), i = 1, \ldots, n$, and compute the nonlinear function $f(x) = \beta_0 + h(x)^T \beta$. The resulting classifier is then $G(x) = sign[f(x)]$.

The solution to the support vector classifier problem (16)-(18) turns out to involve only the inner products of the observations. The inner product of two r-vectors a and b is defined as $\langle a, b\rangle = \sum_{i=1}^{r} a_i b_i$. And the inner product of two observations $x_i, x_{i'}$ is given by

$$\langle x_i, x_{i'}\rangle = \sum_{j=1}^{p} x_{ij} x_{i'j} \tag{22}$$

The support vector classifier can be represented as

$$\begin{aligned} f(x) &= \beta_0 + h(x)^T \beta \\ &= \beta_0 + \sum_{i=1}^{n} \alpha_i y_i \langle h(x), h(x_i)\rangle \end{aligned} \tag{23}$$

where there are n parameters α_i, $i = 1, \ldots, n$, one per training observation. And to estimate the parameters α_i's and β_0, we need the $\binom{n}{2}$ inner products $\langle x_i, x_{i'}\rangle$ between all pairs of training observations. However, α_i happens to be nonzero only for the support vectors in the solution for (23), which means any training observation that is not a support vector has $\alpha_i = 0$. Therefore, to evaluate the function $f(x)$ in (23), we only need to compute the inner product between a new point x and each of the support points x_i. Thus, we can rewrite it as

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i y_i \langle h(x), h(x_i)\rangle \tag{24}$$

Thus, this new function involves far fewer terms and we only need inner products in computing the coefficients. Suppose that we replace the inner product (22) with a generalization of the inner product of the form $K(x_i, xi')$ when it appears in the representation (23), where K is some function called the kernel. It turns out that since (23) involve $h(x)$ only through inner products, we do not need to specify the the transformation $h(x)$ at all. We only need to know the kernel function. A kernel is a function that quantifies the similarity of two observations

$$K(x, x') = \langle h(x), h(x')\rangle \tag{25}$$

which computes inner products in the transformed space, where K is a symmetric positive (semi-) definite function. The advantage of this is that it allows us to construct algorithms in dot product spaces [12]. To fully understand how this works, we need an understanding of kernel functions and the reproducing kernel Hilbert spaces (RKHS), which we suggest other sources for more information [18, 9, 23].

Four popular kernels for SVM are:

$$\text{Linear: } K(x, x') = \langle x, x'\rangle \tag{26}$$

$$d\text{th-Degree Polynomial: } K(x, x') = (1 + \langle x, x'\rangle)^d \tag{27}$$

$$\text{Radial Basis: } K(x, x') = \exp(-\gamma \|x - x'\|^2) \tag{28}$$

$$\text{Neural Network: } K(x, x') = \tanh(\kappa_1 \langle x, x'\rangle + \kappa_2) \tag{29}$$

In the case of the linear kernel, it is simply the support vector classifier. Similarly, the polynomial kernel with degree $d = 1$ is also a support vector classifier. The neural network kernel is also called the sigmoid kernel.

When the support vector classifier is combined with a non-linear kernel, the resulting classifier is called a support vector machine. Therefore, the solution (24) can be rewritten as

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i y_i K(x, x_i) \tag{30}$$

Using kernels allow for the decision boundary to be much more flexible and has a computational advantage. We only need to compute $K(x_i, x_{i'})$ for all $\binom{n}{2}$ distinct pairs i, i' without explicitly working in the enlarged feature space, where the enlarged feature space could be infinite-dimensional. The kernel trick is not a new concept. However, Boser, Guyon, and Vapnik [2] were the first to use the kernel trick with the optimal margin hyperplane to construct a non-linear estimation algorithm to create the support vector machine [12]. Support vector machine is not unique in its use of kernels to enlarge the feature space to accommodate non-linear class boundaries. There are many other classification methods that can use non-linear kernels, such as logistic regression.

1.2.4 SVM Criterion as a Loss Plus Penalty

For fitting the support vector classifier $f(x) = \beta_0 + h(x)^T\beta$, it turns out that the optimization problem (19)-(21) can be rewritten as:

$$\underset{\beta_0, \beta}{\text{minimize}} \sum_{i=1}^{n} [1 - y_i f(x_i)]_+ + \lambda \|\beta\|_2^2 \tag{31}$$

which has the "Loss + Penalty" form

$$L(y, f) + \lambda P(\beta) \tag{32}$$

Here the loss function $L(y, f) = [1 - yf(x)]_+$, where the "+" subscript indicates the positive part, is the hinge loss. It operates on the margin quantity $yf(x)$ and is piecewise linear. The hinge loss is the cost for x_i being on the wrong side of its margin. The cost is 0 if it is on the right side of its margin. The λ is a nonnegative tuning parameter. A large λ has small β and allows more margin violations leading to a classifier that is low variance but high bias. A small λ has large β and allows fewer margin violations leading to a classifier that is high variance but low bias. Thus, we can see that the λ is the cost parameter C in (21). The penalty term in (31), $\lambda\|\beta\|_2^2$, is the ridge penalty term and plays a role in controlling the bias-variance trade-off for the support vector classifier. Therefore, we can see that C is a very important tuning parameter, which determines the extent to which the model is underfitting or overfitting. If the data are separable, then the limit of $\hat{\beta}_\lambda$ in (31) is the optimal separating hyperplane as $\lambda \to 0$. The population minimizer of the hinge loss is the Bayes classifier, and the SVM is directly estimating the classifier $C(x) \in \{-1, +1\}$. The minimizing function for the hinge loss is $f(x) = \text{sign}[\Pr(Y = +1|x) - \frac{1}{2}]$.

The hinge loss function is closely related to the loss function used in logistic regression. Recall that only the support vectors play a role in the support vector classifier and observations on the correct side of the margin do not affect the classifier. This is due to the fact that the hinge loss is exactly zero for observations for which $y_i(\beta_0 + x_i^T\beta) \geq 1$, meaning they are on the correct side of the margin. The loss function for logistic regression is the binomial deviance, $\log[1 + \mathrm{e}^{-yf(x)}]$. The minimizing function for the binomial deviance is the logit $\Pr(Y = +1|x)$. It is not exactly zero anywhere, but it is very small for observations that are far from the decision boundary. For large positive margins, the binomial deviance asymptotes to 0, while for large negative margins, it has a linear loss. The binomial deviance has the same asymptotes as the hinge loss for SVM but is more rounded where near the "elbow" at $yf(x) = 1$. Therefore, logistic regression and the support vector classifier often give very similar results due to the similarities between their loss functions. SVMs tend to have better results than logistic regression when classes are well separated, whereas logistic regression if often preferred with overlapping cases.

1.2.5 Shortcomings

Although support vector machines tend to perform well for large dimensions, it still suffers from the curse of dimensionality. Furthermore, since the algorithm tries to find support vectors using the whole training data to create its classifier, it is time consuming if there is a large number of observations. In addition, like most machine learning algorithms, it is susceptible to imbalanced data and sparse data [4, 1]. Many papers have been published to account for these weaknesses, and more papers are likely to be published with new ideas to extend SVM in these situations.

1.3 Extensions

Since its development 25 years ago, support vector machine drew a huge amount of interest from statistical and non-statistical communities, inspiring thousands of research papers and applications. The support vector machine was initially created for binary classification. However, it has been extended to regression in 1996 by Vapnik and his coworkers, one year after the support vector machine was introduced [8, 19]. There are multi-class extensions of SVMs where there are more than 2 classes [7, 17, 5]. Two well-known methods for multi-class SVM are "one-versus-all" and "one-versus-one" approach. A large number of papers have been published suggesting various extensions to SVMs and multi-class SVMs. Many have been created in an attempt to address the weaknesses of SVM, and many others focus on optimizing computational costs [3]. And a handful of papers were motivated by specific applications ranging from text classification, image and pattern recognition, biomedical and bioinformatic sciences, and many others [21]. Evidently, research on support vector machine is continuously expanding demonstrating its value to the machine learning community. Support vector machines is a powerful and a widely applicable algorithm that has become an essential tool for the machine learning community.

2 Application

To demonstrate the performance of support vector machine, we use the `adult` dataset from the UCI repository [22]. It contains 32,561 observations and 15 variables. Given certain demographic information about an individual, the goal is to predict whether or not an individual earned an income over $50,000. Thus, it is a binary classification.

```r
adult <- read_csv('adult.csv', col_names = c("age", "workclass", "fnlwgt", "education", "education_num"
                                             "marital_status", "occupation", "relationship", "race", "s
                                             "capital_gain", "capital_loss", "hours_per_week",
                                             "native_country", "income"), col_types = cols())
summary(adult)
```

```
      age          workclass              fnlwgt          education        
 Min.   :17.00   Length:32561       Min.   :  12285   Length:32561      
 1st Qu.:28.00   Class :character   1st Qu.: 117827   Class :character  
 Median :37.00   Mode  :character   Median : 178356   Mode  :character  
 Mean   :38.58                      Mean   : 189778                     
 3rd Qu.:48.00                      3rd Qu.: 237051                     
 Max.   :90.00                      Max.   :1484705                     
 education_num   marital_status      occupation        relationship      
 Min.   : 1.00   Length:32561       Length:32561       Length:32561      
 1st Qu.: 9.00   Class :character   Class :character   Class :character  
 Median :10.00   Mode  :character   Mode  :character   Mode  :character  
 Mean   :10.08
 3rd Qu.:12.00
 Max.   :16.00
     race                sex             capital_gain    capital_loss
 Length:32561       Length:32561       Min.   :    0   Min.   :   0.0
 Class :character   Class :character   1st Qu.:    0   1st Qu.:   0.0
 Mode  :character   Mode  :character   Median :    0   Median :   0.0
                                       Mean   : 1078   Mean   :  87.3
                                       3rd Qu.:    0   3rd Qu.:   0.0
                                       Max.   :99999   Max.   :4356.0
 hours_per_week  native_country        income
 Min.   : 1.00   Length:32561       Length:32561
 1st Qu.:40.00   Class :character   Class :character
 Median :40.00   Mode  :character   Mode  :character
 Mean   :40.44
 3rd Qu.:45.00
 Max.   :99.00
```

First, we need to prepare the dataset before applying the algorithm. Observations with missing values coded as "?" are removed. Variables `education` and `education_num` contain the same information but in different formats, so we need only one of them. Furthermore, the distributions of `capital_gain` and `capital_loss` are highly skewed and contain mostly 0's. Therefore, we remove these three variables.

```
adult <- adult %>%
  filter_if(is.character, all_vars(!str_detect(., "\\?"))) %>%
  mutate_if(is.character, as.factor) %>%
  select(-c('capital_gain', 'capital_loss' , 'education_num'))
```

Variables `relationship` and `marital_status` are highly collinear, therefore we remove `relationship`.

```
library(car)
vif(glm(income~., data = adult, family = "binomial"))
```

```
                     GVIF Df GVIF^(1/(2*Df))
age              1.248939  1        1.117559
workclass        1.594731  6        1.039658
fnlwgt           1.049593  1        1.024496
education        2.145596 15        1.025774
marital_status  56.742539  6        1.400100
occupation       2.824897 13        1.040750
relationship   120.530509  5        1.614766
race             3.119605  4        1.152823
sex              2.755820  1        1.660066
hours_per_week   1.135817  1        1.065747
native_country   3.702301 40        1.016497
```

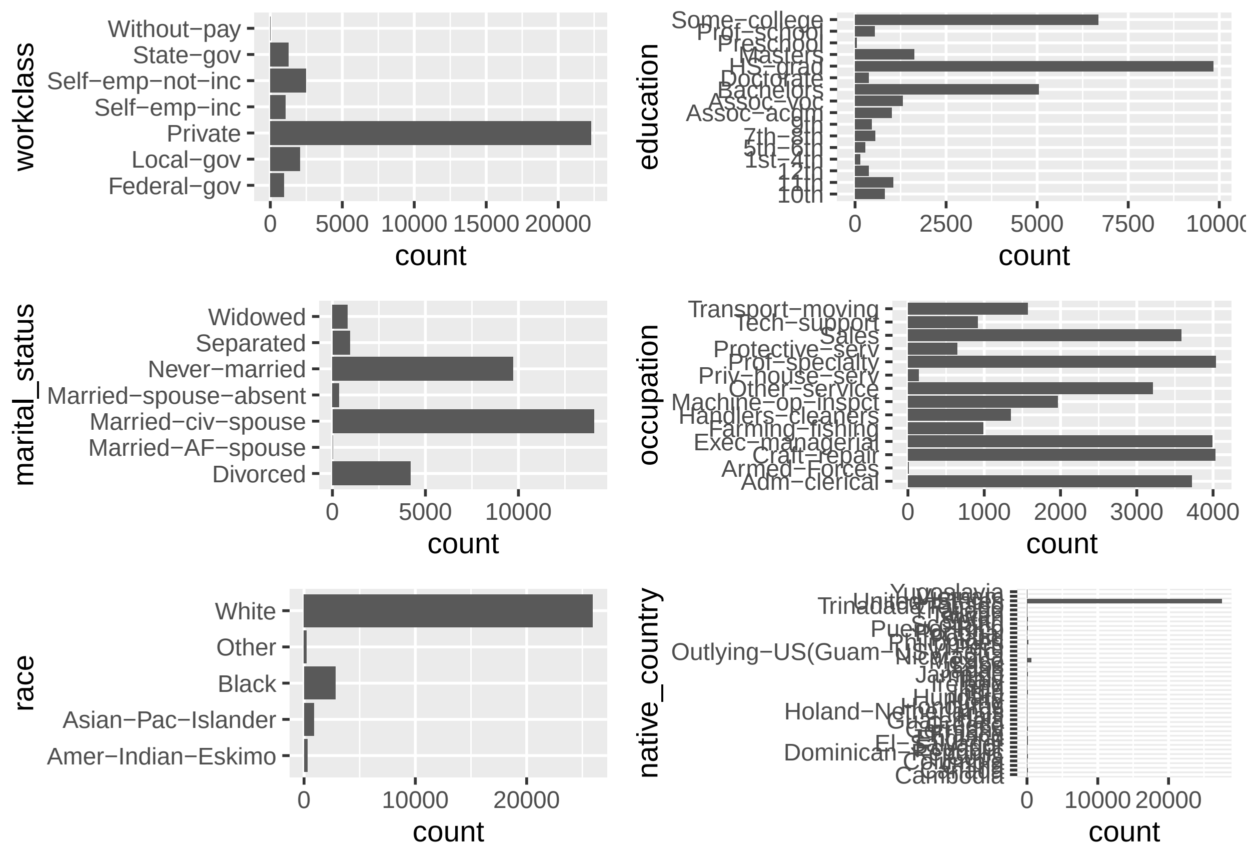
```r
vif(glm(income~.-relationship, data = adult, family = "binomial"))
```

```
                   GVIF Df GVIF^(1/(2*Df))
age            1.236872  1        1.112148
workclass      1.593163  6        1.039573
fnlwgt         1.049787  1        1.024591
education      2.130526 15        1.025533
marital_status 1.571284  6        1.038376
occupation     2.777484 13        1.040072
race           3.085820  4        1.151254
sex            1.454546  1        1.206046
hours_per_week 1.120844  1        1.058699
native_country 3.602477 40        1.016149
```

```r
# Relationship is highly collinear with marital status, thus removed.
adult <- adult %>% select(-relationship)
```

If we examine the class distributions of the categorical variables, we can see that there are too many levels and are unbalanced. For example, `native_country` is almost entirely "United States", so we can combine the rest of the countries as one group. Furthermore, in `workclass`, there is almost no data for `Without-pay` level. This group can mean different things. For example, the person in the `Without-pay` working class can be either be unemployed or working for free such as an internship. Nevertheless, since the person did not technically earn an income, it might not make sense to classify in either income groups. Therefore, it seems practical to remove these observations. Furthermore, we can group the three government levels into one, and the two self-employed working classes into another. Similar logic can be applied to the other categorical variables. For education, we can combine all the groups who did not graduate high school into one group. One might be interested in seeing how pursuing a degree after college can affect income. Therefore, we group people with masters, degrees from professional schools, and doctorates into one group. Occupation tends to be trickier to group. First, "Other service" is very vague and can range from different types of services. Second, one might ask why these specific occupation types were specified. The level of granularity seems to vary between fields. For example, professional specialty can encompass a wide range of occupations, whereas `handler-cleaners` and `priv-house-serv` seem pretty specific. Thus, we only recode two of the occupations since they are very small. Race is recoded as a binary group.

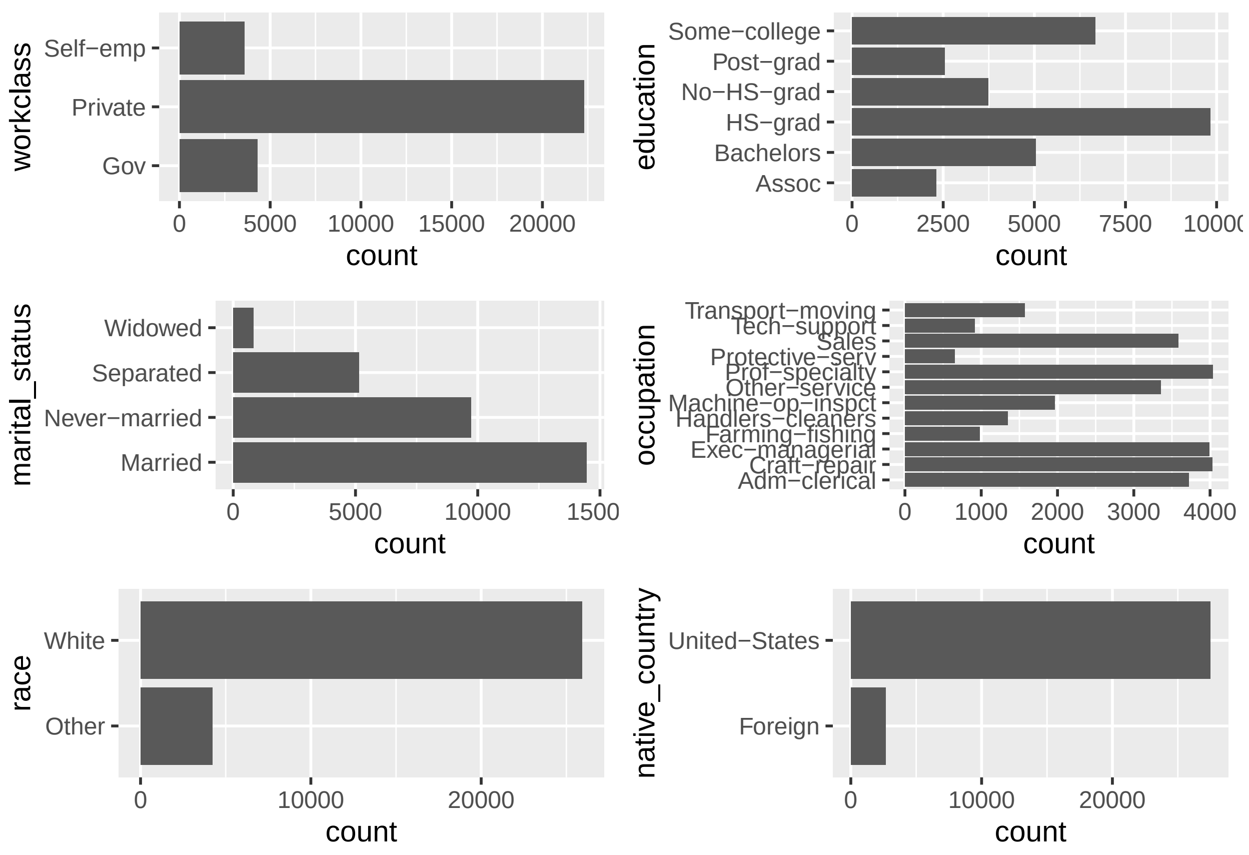
```r
library(miscset)
ggplotGrid(ncol = 2,
           lapply(c("workclass", "education", "marital_status", "occupation", "race", "native_country")
                  function(col) {
                    ggplot(adult, aes_string(col)) + geom_bar() + coord_flip()
                  }))
```



```r
adult <- adult %>% filter(workclass != "Without-pay")
adult$workclass <- droplevels(adult$workclass)
adult$workclass <- recode(adult$workclass, "c('Federal-gov', 'Local-gov', 'State-gov')='Gov';
                          c('Self-emp-inc', 'Self-emp-not-inc')='Self-emp'")
adult$education <- recode(adult$education,
                          "c('Preschool', '1st-4th', '5th-6th', '7th-8th', '9th', '10th', '11th',
                          '12th')='No-HS-grad';
                          c('Prof-school', 'Masters', 'Doctorate')='Post-grad';
                         c('Assoc-acdm', 'Assoc-voc') = 'Assoc'")
adult$marital_status <- recode(adult$marital_status, "c('Divorced', 'Separated')='Separated';
                               c('Married-AF-spouse', 'Married-civ-spouse',
                               'Married-spouse-absent')='Married'")
adult$occupation <- recode(adult$occupation, "c('Armed-Forces')='Protective-serv';
                           'Priv-house-serv'='Other-service'")
adult$race <- recode(adult$race, "c('Amer-Indian-Eskimo', 'Black', 'Asian-Pac-Islander')='Other'")
adult$native_country <- recode(adult$native_country, "'United-States'='United-States'; else='Foreign'")
```

The resulting categorical variables are distributed as below. Although it is still not ideal since there is the imbalanced aspect, it is a considerable improvement.

```r
ggplotGrid(ncol = 2,
           lapply(c("workclass", "education", "marital_status", "occupation", "race", "native_country"),
                  function(col) {
                    ggplot(adult, aes_string(col)) + geom_bar() + coord_flip()
                  }))
```



Now, we will apply the support vector machine algorithm and compare its performance with logistic regression, random forest, k-nearest neighbors, and naive bayes. The `adult` dataset has a large number of observations, which means SVM will suffer as explained previously. Although it is computationally possible, it will be very time consuming to tune the parameters using cross validation with such a big dataset. For the sake of this assignment, we will limit the number of observations to half of the dataset. Randomly sampling half of the dataset results in a similar class distribution for `income` as the original dataset. With this "new" dataset, first, we need to split the data into training and test sets to evaluate the performances on unseen data. Due to the problem of imbalanced data, we use balanced error rate, ROC, and AUC [20]. The balanced error rate is defined as

$$BER = 1 - BCR$$
$$BCR = \frac{1}{2}(TPR + TNR)$$
$$TPR = \frac{TP}{TP + FN}$$
$$TNR = \frac{TN}{TN + FP}$$

where TP is the true positive, FN is the false negative, TN is the true negative, and FP is the false positive. There are other performance metrics that are insenstive to imbalanced data, such as the Geometric Mean, which is also a function of TPR and TNR [20].

```
prop.table(table(adult$income))
```

```
    <=50K      >50K 
0.7509619 0.2490381 
```

```r
# half of original dataset
set.seed(115)
adult <- adult[sample(1:nrow(adult), nrow(adult)*.5),]
prop.table(table(adult$income))
```

```
    <=50K      >50K 
0.7499668 0.2500332 
```

```r
set.seed(123)
index <- sample(1:nrow(adult), nrow(adult)*.7)
train <- adult[index,]
test <- adult[-index, ]

ber <- function(pred, truth) { # balanced misclassification rate
  mat <- table(pred, truth)
  tpr <- mat[2,2]/sum(mat[,2])
  tnr <- mat[1,1]/sum(mat[,1])
  bcr <- (tpr + tnr)/2
  1-bcr
}

library(PRROC)
```

2.1 Support Vector Machine

Both the training data and test data must be scaled before implementing the SVM algorithm. Then, in order to train the SVM algorithm, we need to first decide on a kernel and its respective optimal parameters. In the guide for SVM written by the creators of `LIBSVM`, the authors suggest first using the radial basis function (RBF) kernel, also known as radial kernel for three reasons [13]. The linear kernel and sigmoid kernel could have similar performances to the radial kernel based on their respective parameters, it has less parameters to optimize than the polynomial kernel, and easier to implement numerically than some other kernels. And since the number of variables do not exceed the number of observations, we can use the radial kernel rather than the linear kernel.

The radial kernel has two parameters that need to be optimized, C and γ. The `caret` package can do this automatically by finding the best parameters using 10-fold cross validation [16]. We use ROC to find the best parameters.

```r
Train <- train
Train$income <- as.factor(ifelse(train$income == "<=50K", 'LT50', 'GT50')) # format for caret
library(caret)
library(e1071)
library(kernlab)
library(doParallel)
cl <- makeForkCluster(detectCores()-2)
registerDoParallel(cl)
ctrl <- trainControl(method = "cv",
                     number = 10,
                     classProbs = TRUE,
                     allowParallel = TRUE)
svmfit <- train(income ~ .,
                      data = Train,
                      method = "svmRadial",
                      preProcess = c('center', 'scale'),
                      metric = "Kappa",
                      trControl = ctrl,
                      tuneLength = 10)
stopCluster(cl)

svmfit
```

```
 Support Vector Machines with Radial Basis Function Kernel

10551 samples
   10 predictor
    2 classes: 'GT50', 'LT50'

Pre-processing: centered (27), scaled (27)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 9496, 9495, 9496, 9497, 9496, 9495, ...
Resampling results across tuning parameters:

  C       Accuracy   Kappa
    0.25  0.8230466  0.4765984
    0.50  0.8236139  0.4770035
    1.00  0.8231408  0.4752251
    2.00  0.8238990  0.4784283
    4.00  0.8222880  0.4762326
    8.00  0.8217197  0.4769297
   16.00  0.8217204  0.4815746
   32.00  0.8207728  0.4788456
   64.00  0.8157488  0.4611399
  128.00  0.8091139  0.4371216

Tuning parameter 'sigma' was held constant at a value of 0.0241029
Kappa was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.0241029 and C = 16.
```

The best parameters that the model returned are 0.024, 16.

Now we classify our test data using the best parameters obtained from our training data.

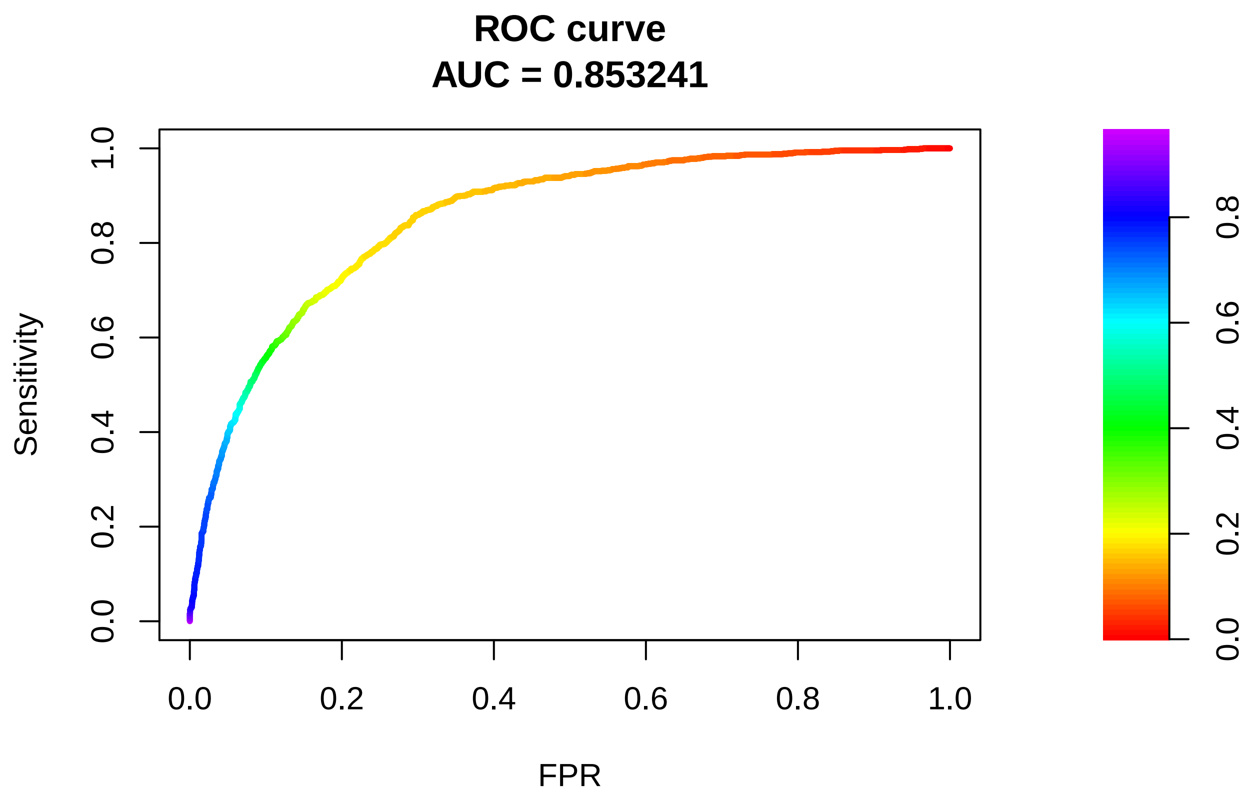
```
pred <- predict(svmfit, test, "prob")
pred.svm <- ifelse(pred[,1] >= 0.5, ">50K", "<=50K")
table(pred.svm, test$income)
```

```
pred.svm <=50K >50K
   <=50K  3109  564
   >50K    271  579
```

```
(svm.ber <- ber(pred.svm, test$income))
```

```
## [1] 0.2868079
```

```
y <- ifelse(test$income %in% "<=50K", 0, 1)
svm.auc <- roc.curve(scores.class0 = pred[,1], weights.class0 = y,
                      curve=TRUE)
plot(svm.auc)
```



The balanced misclassification rate is 29%.

2.2 Comparison

We compare support vector machine with 4 other classification methods: logistic regression, random forest, KNN, and Naive Bayes.

2.2.1 Logistic Regression

Luckily, with logistic regression, there is no need for data pre-processing, unlike SVM.

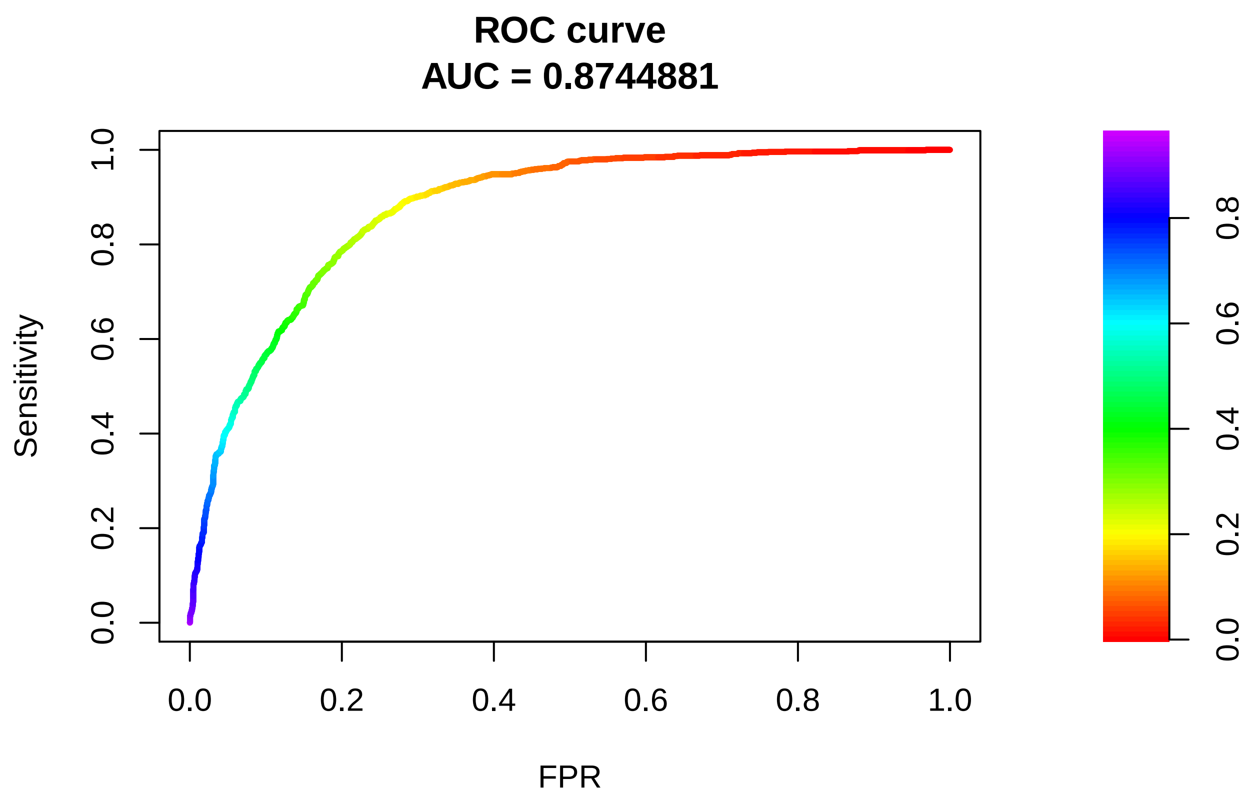
```
logfit <- glm(income ~., family = "binomial", data = train)
pred <- predict(logfit, newdata = test, type = "response")
pred.log <- ifelse(pred > .5, ">50K", "<=50K")
table(pred.log, test$income)
```

```
pred.log <=50K >50K
   <=50K  3108  561
   >50K    272  582
```

```
(log.ber <- ber(pred.log, test$income))
```

```
 [1] 0.2856435
```

```
log.auc <- roc.curve(scores.class0 = pred, weights.class0 = y,
                       curve=TRUE)
plot(log.auc)
```



Logistic regression gives us a balanced misclassification rate of 29%.

2.2.2 Random Forest

For random forest, we first tune the parameter mtry, which is the number of variables to randomly select, using 10-fold cross validation. Then, we classify the test data using the model.

```
cl <- makeForkCluster(detectCores()-2)
registerDoParallel(cl)
rffit <- train(income ~ .,
                      data = Train,
                      method = "rf",
                      metric = "Kappa",
                      trControl = trainControl(method = "cv", number = 10, allowParallel = TRUE),
                      tuneLength = 20)
stopCluster(cl)
pred <- predict(rffit, test, type = "prob")
pred.rf <- ifelse(pred[,1] > .5, ">50K", "<=50K")
table(pred.rf, test$income)
```

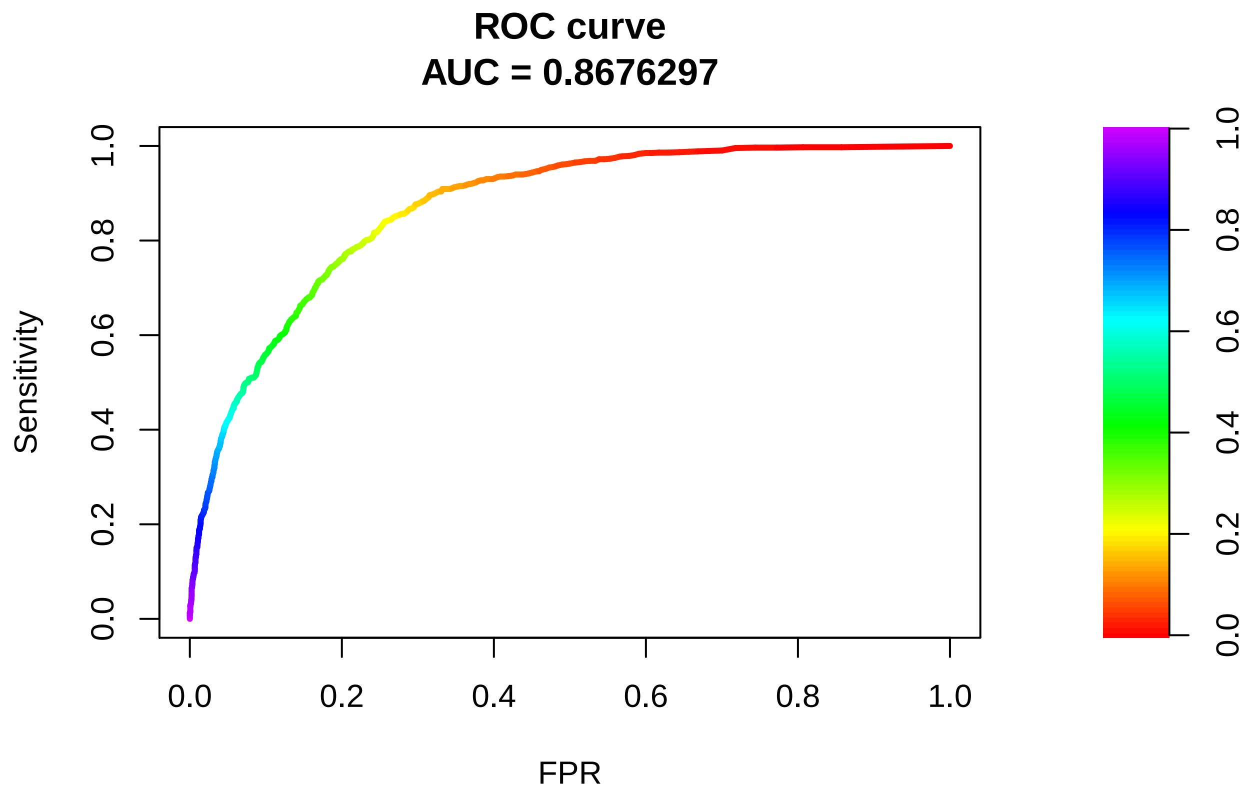
```
pred.rf <=50K >50K
  <=50K  3084  550
  >50K    296  593
```

```

```

```r
(rf.ber <- ber(pred.rf, test$income))
```

```
[1] 0.2843819
```

```r
rf.auc <- roc.curve(scores.class0 = pred[,1], weights.class0 = y,
                    curve=TRUE)
plot(rf.auc)
```



Random forest gives us 28% balanced misclassification rate.

2.2.3 K-Nearest Neighbors

For KNN, we need to make sure the data scales from 0 to 1. Then, we use 10-fold cross validation to find the tuning parameter k, the number of neighbors. Using the training model, we classify the test data.

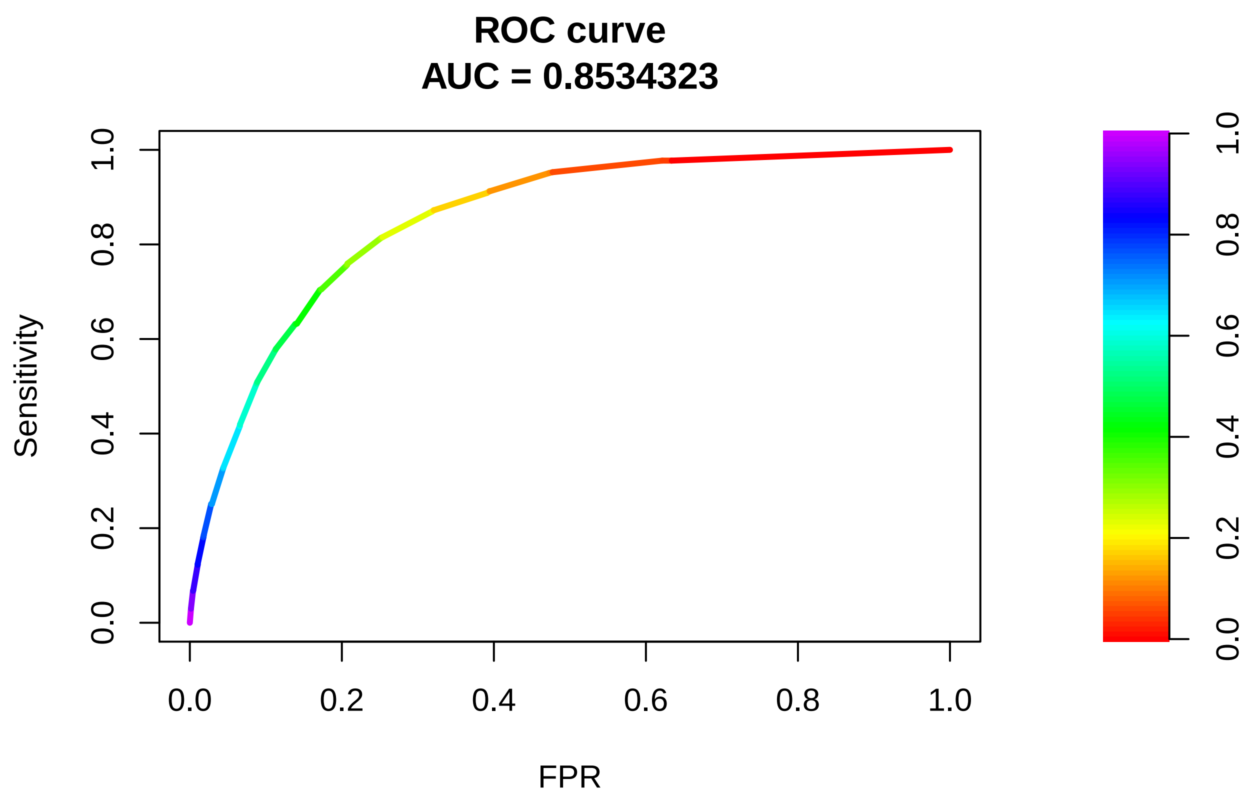
```r
cl <- makeForkCluster(detectCores()-2)
registerDoParallel(cl)
knnfit <- train(income ~ .,
                data = Train,
                method = "knn",
                preProcess = "range",
                metric = "Kappa",
                trControl = ctrl,
                tuneLength = 10)
stopCluster(cl)

pred <- predict(knnfit, test, type = "prob")
pred.knn <- ifelse(pred[,1] > .5, ">50K", "<=50K")
table(pred.knn, test$income)
```

```
pred.knn <=50K >50K
   <=50K  2999  483
   >50K    381  660
```

```r
(knn.ber <- ber(pred.knn, test$income))
```

```
 [1] 0.267647
```

```r
knn.auc <- roc.curve(scores.class0 = pred[,1], weights.class0 = y,
                     curve=TRUE)
plot(knn.auc)
```



K-nearest neighbors gives us 27% balanced misclassification rate.

2.2.4 Naive Bayes

Now we compare Naive Bayes with SVM.

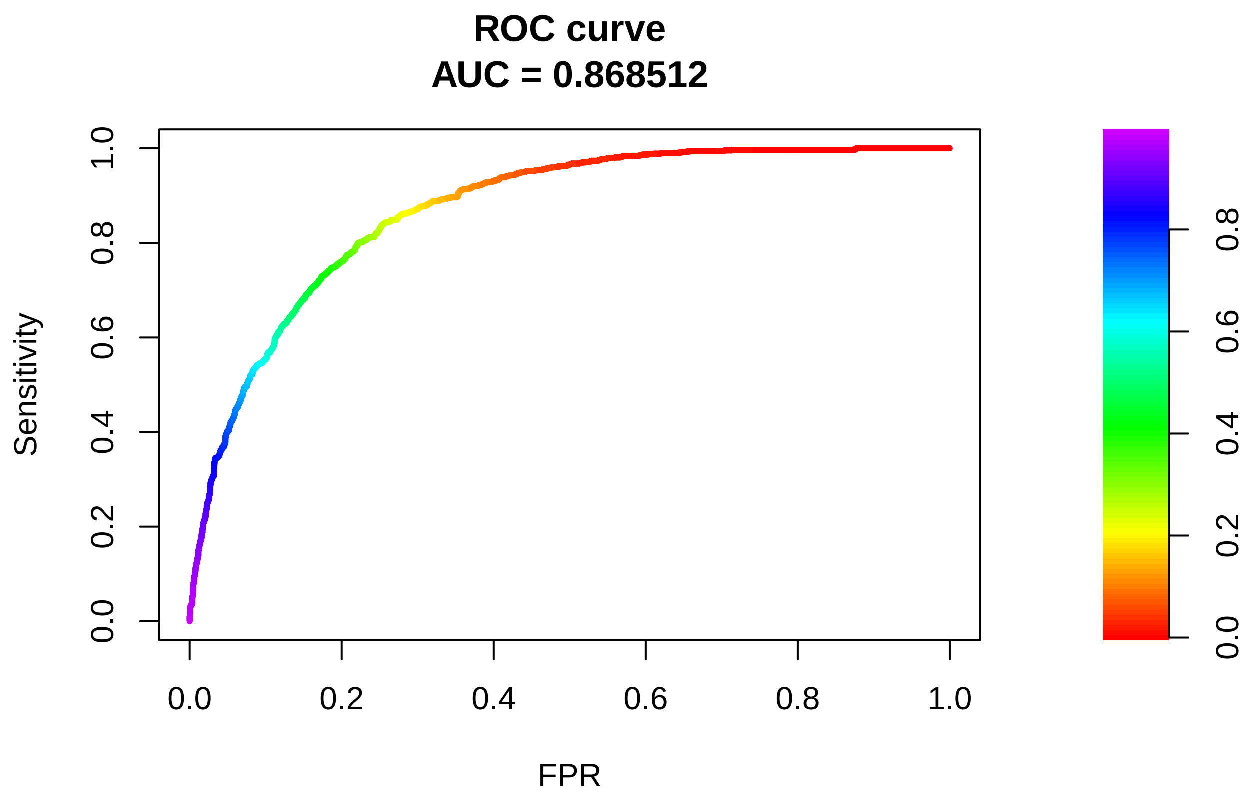
```
library(naivebayes)
nbfit <- naive_bayes(income~., train, laplace = 0.5)
pred.nb <- nbfit %class% test[,-11]
pred <- predict(nbfit, test[,-11], 'prob')
table(pred.nb, test$income)
```

```
pred.nb <=50K >50K
  <=50K  2915  395
  >50K    465  748
```

```
(nb.ber <- ber(pred.nb, test$income))
```

```
[1] 0.2415779
```

```
nb.auc <- roc.curve(scores.class0 = pred[,2], weights.class0=y,
                    curve=TRUE)
plot(nb.auc)
```



Naive Bayes gives us 24% balanced misclassification rate.

2.3 Conclusion

Overall, all five classification methods gave similar results. Because we are training the classification methods, except logistic regression, in parallel, without setting seeds, the results tend to differ, where logistic regression outperforms SVM and the other methods, or Naive Bayes outperforms others. However, each time, SVM is still pretty competitive with the other classification methods. The weakness to SVM is the number of observations. With just half of the original dataset, it still took longer to find the support vectors after using cross-validation and grid search to fine tune the parameters, compared to the other methods. In addition, we have the problem of imbalanced data which is another weakness of SVM. One method to address this is to change the data structure where the imbalanced dataset is balanced by oversampling the minority class, undersampling the majority class, or synthetic minority oversampling technique (SMOTE) [1, 15]. Another is a cost-sensitive method where different weights are assigned to the misclassification costs of the majority and minority classes. Both approaches have drawbacks, and the research on tackling imbalanced dataset is ongoing.

Although we haven't explored these methods of addressing imbalanced data, it would be interesting to see how competitive these algorithms are when imbalance is taken into account. Imbalanced data is a continuous problem for most, if not all, machine learning algorithms, including SVM, and have been the subject of many papers and articles [15, 1]. In addition, we can see how SVM can suffer if the number of observations far exceed the number of features. Regardless, SVM has shown to be immensely powerful, universal, and flexible as Vapnik and Cortes claimed 25 years prior. The sheer number of research papers and search results suggest the importance and value of support vector machines in the machine learning community as well as the statistical community.

References

[1] Rukshan Batuwita and Vasile Palade. "Class Imbalance Learning Methods For Support Vector Machines". In: (2012).

[2] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. "A Training Algorithm for Optimal Margin Classifiers". In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT '92. New York, NY, USA: Association for Computing Machinery, July 1, 1992. ISBN: 978-0-89791-497-0. DOI: 10.1145/130385.130401.

[3] Christopher J C Burges and Bernhard Schölkopf. "Improving the Accuracy and Speed of Support Vector Machines". In: (1997).

[4] Jair Cervantes et al. "A Comprehensive Survey on Support Vector Machine Classification: Applications, Challenges and Trends". In: (2020).

[5] Chih-Wei Hsu and Chih-Jen Lin. "A Comparison of Methods for Multiclass Support Vector Machines". In: *IEEE Transactions on Neural Networks* 13.2 (Mar. 2002). DOI: 10.1109/72.991427.

[6] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Machine Learning* 20.3 (Sept. 1995). DOI: 10.1007/BF00994018.

[7] Koby Crammer and Yoram Singer. "On the Algorithmic Implementation of Multiclass Kernel-Based Vector Machines". In: (2001).

[8] Harris Drucker et al. "Support Vector Regression Machines". In: (1996).

[9] Bradley Efron and Trevor Hastie. *Computer Age Statistical Inference*. 2016.

[10] R. A. Fisher. "The Use of Multiple Measurements in Taxonomic Problems". In: *Annals of Eugenics* 7.2 (Sept. 1936). DOI: 10.1111/j.1469-1809.1936.tb02137.x.

[11] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY: Springer New York, 2009. ISBN: 978-0-387-84857-0 978-0-387-84858-7. DOI: 10.1007/978-0-387-84858-7.

[12] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. "Kernel Methods in Machine Learning". In: *The Annals of Statistics* 36.3 (June 2008). DOI: 10.1214/009053607000000677. arXiv: math/0701907.

[13] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. "A Practical Guide to Support Vector Classification". In: (2016).

[14] Gareth James et al. *An Introduction to Statistical Learning*. Vol. 103. Springer Texts in Statistics. New York, NY: Springer New York, 2013. ISBN: 978-1-4614-7137-0 978-1-4614-7138-7. DOI: 10.1007/978-1-4614-7138-7.

[15] Bartosz Krawczyk. "Learning from Imbalanced Data: Open Challenges and Future Directions". In: *Progress in Artificial Intelligence* 5.4 (Nov. 2016). DOI: 10.1007/s13748-016-0094-0.

[16] Max Kuhn. *The Caret Package*. 2019. URL: https://topepo.github.io/caret/index.html.

[17] John C Platt, Nello Cristianini, and John Shawe-Taylor. "Large Margin DAGs for Multiclass Classification". In: (2000).

[18] Bernhard Schölkopf and Alexander J. Smola. “A Short Introduction to Learning with Kernels”. In: *In Advanced Lectures on Machine Learning, S.Mendelson*. Springer, 2002.

[19] Alex J. Smola and Bernhard Schölkopf. “A Tutorial on Support Vector Regression”. In: *Statistics and Computing* 14.3 (Aug. 2004). DOI: 10.1023/B:STCO.0000035301.49549.88.

[20] Alaa Tharwat. “Classification Assessment Methods”. In: *Applied Computing and Informatics* ahead-of-print (ahead-of-print Aug. 3, 2020). DOI: 10.1016/j.aci.2018.08.003.

[21] Simon Tong and Daphne Koller. “Support Vector Machine Active Learning with Applications to Text Classification”. In: (2001).

[22] *UCI Machine Learning Repository: Adult Data Set*. URL: https://archive.ics.uci.edu/ml/datasets/Adult.

[23] Grace Wahba. *Support Vector Machines, Reproducing Kernel Hilbert Spaces and the Randomized GACV*. 1998.