FYS3150 Machine Learning - Project 3
Classifying Pulsar Candidates With Machine Learning Methods

Nils Johannes Mikkelsen
(Dated: December 2, 2018)

All material for project 3 may be found at:
https://github.com/njmikkelsen/machinelearning2018/tree/master/Project3

I. INTRODUCTION

II. THEORY

A. Radio Analysis of Pulsars

The following theory is mostly based on F.G. Smith's 1977 *Pulsars* [3], with additional information taken from "Chapter 6: Pulsars" of Jim Condon's online book *Essential Radio Astronomy* [2].

1. *The basics of pulsars*

Pulsars are heavily magnetised rotating astronomical objects that emit regular pulses of electromagnetic (EM) radiation. Initially discovered in 1967 by Antony Hewish and Jocelyn Bell Burnell, it is generally believed by most modern astronomers that pulsars are neutron stars, the product of some supernovae. There are several mechanisms that result in the electromagnetic radiation (although still not entirely understood), astronomers have thus devised three major categories of pulsars according to the source of radiation: rotation-powered pulsars, accretion-powered pulsars and magnetars. Rotation-powered pulsars use the loss of rotational energy, accretion-powered pulsars exploit the gravitational potential energy of accreted matter, while magnetars are believed to power its radiation pulses by the decay of an extremely strong magnetic field ($\sim 10^{10}$ T). Other than regular pulses of radiation, the properties of the radiation from the different categories are mostly different. Rotation-powered pulsars radiate mostly in the Radio spectrum, while the accretion-powered pulsars radiate mostly in the X-Ray spectrum. Magnetars have been observed to radiate both in the X-Ray and gamma-ray spectra.

The data studied in this project stem from radio-telescope observations, thus only rotation-powered pulsars are considered here.

2. *ISM dispersion*

As the name implies, radio pulsars are particular pulsars that radiate in the radio spectrum. When astronomers first began studying radio pulsars they noticed that radiation from the same pulsar with different frequencies arrived on Earth at different times. It was quickly pointed out that this EM dispersion coincided with the expected dispersion of EM waves travelling through the cold plasma of the interstellar medium (ISM). Travelling through an ionised gas, the group and phase velocities of radio waves, v_g and v_p respectively, are related by

$$v_g v_p = c^2 \tag{1}$$

where c is the speed of light in a vacuum. The fractive index of the cold plasma is

$$\mu = \sqrt{1 - \left(\frac{\nu_\mathrm{p}}{\nu}\right)^2} \tag{2}$$

where ν is the radio wave frequency and ν_p is the *plasma frequency* given by:

$$\nu_\mathrm{p} = \sqrt{\frac{e^2 n_e}{\pi m_e}} \tag{3}$$

Here, n_e and m_e are the electron number density and m_e is the electron mass. Now, radio waves with $\nu \leq \nu_\mathrm{p}$ are unable to propegate through the ISM. On the other hand, waves with $\nu > \nu_\mathrm{p}$ do propegate through the ISM with a group velocity of

$$v_g = \mu c \cong \left(1 - \frac{\nu_\mathrm{p}^2}{2\nu^2}\right) c \tag{4}$$

Hence, the travel time T spent by a radio pulse from a pulsar a distance L away is

$$\begin{aligned} T &= \int_0^L \frac{1}{v_g}\,\mathrm{d}l = \frac{1}{c}\int_0^L \left(1 + \frac{\nu_\mathrm{p}^2}{2\nu}\right)\mathrm{d}l \\ &= \frac{d}{c} + \left(\frac{e^2}{2\pi m_e c}\right)\nu^{-2}\int_0^L n_e\,\mathrm{d}l \end{aligned} \tag{5}$$

The integral in (5) is commonly called the *Dispersion Measure*, and is commonly given in units of pc cm^{-3}:

$$\mathrm{DM} = \int_0^L n_e\,\mathrm{d}l \tag{6}$$

The Dispersion Measure is not a trivial quantity to estimate because it depends on the constituents and their arrangements of the ISM, which in general is not uniform. Despite accurate theoretical predictions of DM, it continues to be an essential variable parameter in radio astronomy analysis, more or this below. In astronomy-convenient units, the dispersion delay $t = T - d/c$ due to the ISM is

$$t \cong 4.149 \cdot 10^3 \nu^{-2}\,\mathrm{DM}\text{ seconds} \tag{7}$$

where ν in MHz and DM in pc cm^{-3}.

3. *The integrated pulse profile*

Due to difficulties with time-resolution, limited bandwidth, etc., studying a single radio pulse is considerably more difficult than to study the "average pulse" across several frequencies. Taking into account the time delay due to dispersion in the ISM (equation (7)), the radio pulses from radio pulsars may be *folded* to yield an integrated pulse profile. Seeing that the DM is essential in the computation of the time delay, astronomers often study the integrated pulse profile as a function of DM. An example of the folding process is shown in figure 1, note that the delay folds across several pulse periods. Moreover, while the time-signal is fairly chaotic, the integrated pulse profile is very focused.

While all pulsars share the property of a stable pulse profile, it turns out that each pulsar exhibits a particular profile that is more or less unique. This unique profile, often nicknamed the "fingerprint" of the pulsar, can have several peaks and be less or more spread out. Nonetheless, some statistical properties such as a single peak, a double peak, various minor peaks, etc., do occur quite frequently.

4. *Signal-to-noise ratio*

An important quantity in observational astronomy is the Signal-to-Noise Ratio (SNR), which in its simplest form is defined as the ratio of the power of the signal to the power of the noise:

$$\mathrm{SNR} = \frac{P_{\text{signal}}}{P_{\text{noise}}} \tag{8}$$

SNR is a measure of the quality of observation, or "amount of meaningful information" in an observation. Astronomers actually use the SNR to study the DM parameter space, as selecting the optimal DM is essential for conducting a proper analysis of the integrated pulse profile. Enter the the DM-SNR curve: the SNR score of an integrated pulse profile as a function of DM. The DM-SNR curve is usually flat for non-pulsating signals, however peaks about the optimal DM if the signal does pulsate. Consequently, the DM-SNR curve is commonly used as a preliminary test for identifying signals from pulsars in large data sets of radio signals.

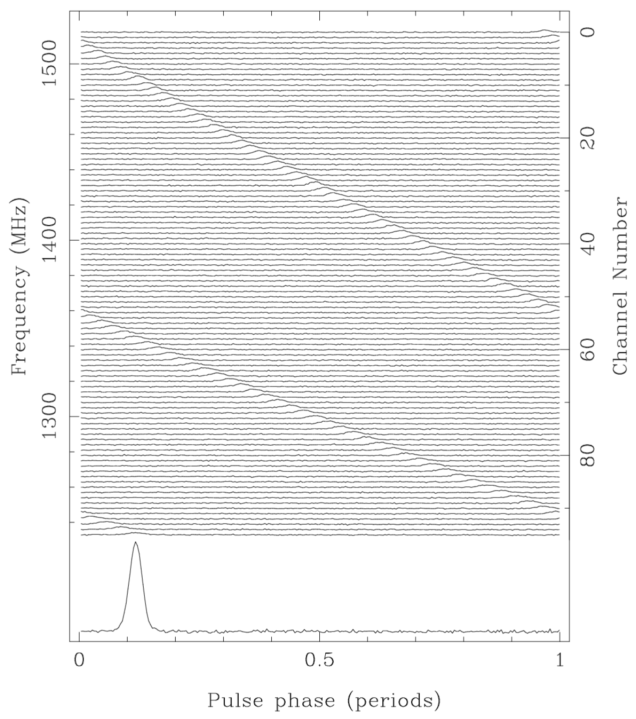

Figure 1: The pulse profile of a radio wave experiences a time delay due to dispersion in the ISM. The integrated pulse profile is a folded profile that finds an average pulse profile by taking into account the time delay. Source: [1]

B. Moments In Statistics

Given some continuous function $f : \mathbb{R} \to \mathbb{R}$, its n^{th} *moment* $\mu_n(c)$ about the real value c is defined as

$$\mu_n(c) = \int_{-\infty}^{\infty} (x - c)^n f(x)\,\mathrm{d}x \tag{9}$$

Moments are used in several areas of science and mathematics, most prominently in mechanics and probability theory. If f is a probability density (or strictly non-negative and normalizable),[1] there exists two major notation conventions:

$$\mu_n(c) = E\big[(X - c)^n\big] \quad \text{and} \quad \mu_n(c) = \langle (x - c)^n \rangle$$

The former is favoured by statisticians, while the latter is favoured by physicists, this project will employ the physicists' convention.

[1] A probability density $f : D \to \mathbb{R}$ satifies

$$0 \le f(x),\ \forall x \in D \quad \text{and} \quad \int_D f(x)\,\mathrm{d}x = 1.$$

Note that D is usually the entire real number line.

The moments most often studied in probability theory are the so-called *central* moments μ_n:

$$\mu_n = \mu_n(\mu) = \langle (x-\mu)^n \rangle = \int_{-\infty}^{\infty} (x-\mu)^n f(x)\,\mathrm{d}x \quad (10)$$

where $\mu = \langle x \rangle$ is the mean value of the distribution and $\mu_n = \mu_n(\mu)$ is introduced to simplify notation. The 0^{th} and 1^{st} central moments are 1 and 0 respectively as $f(x)$ is normalized and $(x-\mu)f(x)$ is centered about μ. The 2^{nd} central moment is the variance:

$$\mu_2 = \left\langle (x-\mu)^2 \right\rangle = \sigma^2 \quad (11)$$

where σ is the standard deviation.

Furthermore, the central moments are used to define the *normalized* moments $\tilde{\mu}_n$:

$$\tilde{\mu}_n = \frac{\mu_n}{\sigma^n} = \frac{\langle (x-\mu)^n \rangle}{\left\langle (x-\mu)^2 \right\rangle^{n/2}} \quad (12)$$

Normalized moments are dimensionless quantities that describe the underlying distribution independently of the scale of the data. The 3^{rd} normalized moment is known as the *skewness* of a distribution and quantifies the extent to which a distribution is *skewed* to "left or right" of the mean. Positive skewness implies the distribution is skewed towards x values greater than μ and vice versa. The 4^{th} normalized moment is called *kurtosis* and is a measure of how quickly the tails of a distribution tends to zero, i.e. a measure of "*peakness*". Kurtosis is usually reported as *excess kurtosis*, which is the kurtosis of a distribution relative to the normal distribution (whose kurotsis is 3). Hence the definition:

$$\text{Excess Kurtosis} \;=\; \text{Kurtosis} - 3 \quad (13)$$

C. Machine Learning - Artificial Neural Networks

Artificial Nerual Networks (ANN or simply NN) is a Supervised Machine Learning (SML) technique that is inspired by biological neural networks in animal brains. Rather than a specific technique or algorithm, ANN is an umbrella term for several NN structures and algorithms whose complexity, computational difficulty, etc., varies greatly between different techniques.

1. *The Multi-Layered Perceptron*

This project will employ the so-called Multi-Layered Perceptron (MLP) model, which is a particular ANN that is based around the idea of organizing the network's neurons, or nodes, in layers. A node is loosely based on the concept of biological neuron, which is capable of recieving an electrical signal, process it, and transmit a new signal to other neurons. In the MLP model the neurons are arranged in L consecutive layers with N_l number of nodes per layer (unqeual number of nodes per layer is alloweed). There are three types of layers: input, hidden and output layers. The input and output layers manage the network's inputs and outputs, while the hidden layers serve as the network's data-processing backbone. Particular to the MLP, every node in each hidden layer is connected to every node in the previous and the next layers, but nodes within the same layer are *not* connected. The MLP structure is illustrated in figure 2, note the inclusion of "bias": Bias is an additional property of all layers except the input layer whose purpose is essentially to center the data about 0. Conceptually, one may think of bias as the role of the intercept in Linear Regression analysis.

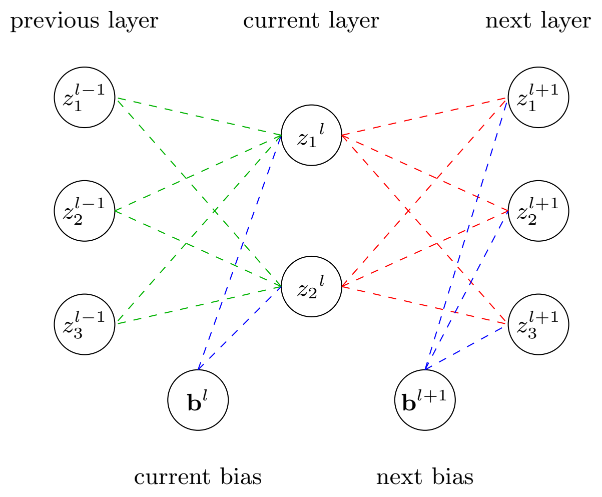

Figure 2: An illustration of the structure of an MLP ANN: Each node is connected to every node in the previous layer and the next layer. The inputs to the current layer consists of the data from the previous layer (green connections) and the current layer's bias (left blue connections). The outputs of the current layer (red connections) together with the next layer's bias (right blue connections) become the inputs of the next layer, and so on.

So far the network has only been described using fluid terms such as "transfer of information" and "node connections", this will be expanded upon now. The number of nodes in the input layer N_1 must exactly match the dimensionality of the input data, thus there is a one-to-one correspondence between the nodes and the indices of the input data (usually a vector quantity). Similarly, the number of nodes in output layer N_L must exactly match the output data of the network, again with a one-to-one correspondence between nodes and data indices. It follows that in practice it is really only the hidden layers that may have a variable number of nodes N_l, $l \in \{2, \ldots, L-1\}$. The network processes input data layer-by-layer, meaning each hidden layer processes the data in sequence. Data from the input layer is sent to the first hidden layer, processed, then transfered to the next layer, and so on.

Mathematically speaking, the connection between two nodes i in layer l and j in layer $l+1$ is a scaling operation. The connection's scaling factor is usually called the weight and is commonly denoted by w_{ij}^{l+1}, note the convention of labeling a connection weight between layers l and $l+1$ by $l+1$. Each node is connected to every node in the previous layer, thus the purpose of weighting the data is to regulate the importance of each node in the previous layer. However, note that unless some other operation is conducted on the data the network's overall operation on the input data is only linear (meaning methods such as Linear Regression are more appropriate). This is circumvented by the addition of a non-linear operation on the every node's input, commonly called the node's *activation* function. Common choices of activations are the sigmoid function and the tangent hyperbolicus:

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{14a}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{14b}$$

Other activation functions are also used. Technically, a network may have a different activation for each node, but this is rarely done in practice. On the other hand, layer-wide activation functions are often implemented, so much so that the following theory will base the mathematics on the presumption of layer-wide activation.

The data in node i in layer l is denoted by y_i^l. The connection weight between node i in layer l and node j in layer $l+1$ is denoted by w_{ij}^{l+1}. The accumulated data input from all nodes in layer l received by node j in layer $l+1$, adjusted for bias b_j, is given by

$$z_j^{l+1} = \sum_{i=1}^{N_l} w_{ij}^{l+1} y_i^l + b_i^l \tag{15}$$

The input data received by the node is $f_{l+1}(z_j^{l+1})$, where f_{l+1} is the activation function of layer $l+1$. Hence, the data in node j in layer $l+1$ is given by

$$y_j^{l+1} = f_{l+1}(z_j^{l+1}) = f_{l+1}\left(\sum_{i=1}^{N_l} w_{ij}^{l+1} y_i^l + b_j^{l+1}\right) \tag{16}$$

Arranging the node data in vectors $\mathbf{y}^l$, the node inputs in vectors $\mathbf{z}^{l+1}$, the connection weights in a weight matrix $\hat{W}^{l+1}$ and the bias in bias vectors $\mathbf{b}^{l+1}$, then (16) can be reformulated as a layer-wide (matrix) operation:

$$\mathbf{y}^{l+1} = f_{l+1}(\mathbf{z}^{l+1}) = f_{l+1}\left(\hat{W}^{l+1}\mathbf{z}^l + \mathbf{b}^{l+1}\right) \tag{17}$$

To conclude this section: The answer to the question *"What is a Multi-Layered Perceptron Neural Network?"* is the following function:

$$\mathbf{y} = f_L\left(\hat{W}^L f_{L-1}\left(\cdots\left(\hat{W}^1\mathbf{x} + \mathbf{b}^1\right)\cdots\right) + \mathbf{b}^L\right) \tag{18}$$

where $\mathbf{x} = \mathbf{y}^1$ and $\mathbf{y} = \mathbf{y}^L$ are the network's inputs and outputs.

2. Training an MLP

The basis of the MLP, and many similar network structures, is the so-called *back-propagation* algorithm (BPA). The MLP is an example of a *supervised* machine learning technique, meaning the desired outputs are known. In case of the MLP, it is the BPA that introduces supervision.

At its core, the BPA is essentially a variant of gradient descent techniques. The basic problem of gradient descent methods is to find the input $\mathbf{x}_{\min} \in D$ that minimises some function $F : D \to \mathbb{R}$. The algorithm is based on producing sucessive approximations of $\mathbf{x}_{\min}$, each closer to the true minium. Seeing that the goal is to minimise F, the direction of each step is chosen opposite of the gradient of F. The fundamental algorithm is summarised below:

Algorithm 1 Gradient Descent

```
1: Select a trial input $\mathbf{x}_0$.
2: for $i = 1, \ldots, N$:
3:     Compute: $\mathbf{x}_i = \mathbf{x}_{i-1} - \gamma_n \frac{\partial F}{\partial \mathbf{x}}\Big|_{\mathbf{x}_{i-1}}$
4: end for
```

Note that the derivative is not scalar. The step length γ_n may be variable or constant depending on the implementation, although proper convergence is impossible with too large steps.

The F function to minimise in the BPA is the so-called *cost* function, denoted C. Similar to activation functions, different cost functions are used in different problems. The cost function most used in classification analysis is the so-called cross-entropy:

$$C(\mathbf{y}|\mathbf{t}) = -\Big[\mathbf{t}^T \ln(\mathbf{y}) + (1-\mathbf{t})^T \ln(1-\mathbf{y})\Big] \tag{19}$$

where $\mathbf{y}$ is a vector of the network's outputs and $\mathbf{t}$ is a vector of the targets, i.e. the desired network output.[2] Note that $C(\mathbf{y}|\mathbf{t})$ is univariate: it is a function of outputs $\mathbf{y}$ provided the targets $\mathbf{t}$ (that is, $\mathbf{t}$ is not a variable quantity). The derivative of $C(\mathbf{y}|\mathbf{t})$ is

$$\frac{\partial C}{\partial \mathbf{y}} = (\mathbf{y} - \mathbf{t}) \oslash \Big(\mathbf{y} \circ (1-\mathbf{y})\Big) \tag{20}$$

where $\oslash$ and $\circ$ denotes the Hadamard division and product (element-wise operations). Now, if this was a problem for gradient descent methods, then $\mathbf{y}$ would be updated directly using Algorithm 1 and equation (20). However, $\mathbf{y}$ is produced by the network according to equation (18),

[2] The introduction of $\mathbf{t}$ in the cost function is the actual introduction of machine learning supervision in MLP ANNs. Also, note that naming $\mathbf{t}$ the "targets" refers to the goal of neural networks: namely to construct the network such that it reproduces the target outputs.

which is dependent on the weights and biases of each layer. Hence, in order for the BPA to improve the network it must study $C(\mathbf{y}|\mathbf{t})$ with respect to all weights and biases.

Although seemingly difficult, the problem is actually somewhat straight-forward, it all comes down to clever use of the chain rule and limiting the number of necessary computations. Observe that the derivative of $C(\mathbf{y}|\mathbf{t})$ with respect to layer l can be written as

$$\frac{\partial C}{\partial \hat{W}^l} = \frac{\partial C}{\partial \mathbf{y}^l}\frac{\partial \mathbf{y}^l}{\partial \mathbf{z}^l}\frac{\partial \mathbf{z}^l}{\partial \hat{W}^l} \tag{21a}$$

$$\frac{\partial C}{\partial \mathbf{b}^l} = \frac{\partial C}{\partial \mathbf{y}^l}\frac{\partial \mathbf{y}^l}{\partial \mathbf{z}^l}\frac{\partial \mathbf{z}^l}{\partial \mathbf{b}^l} \tag{21b}$$

Now define the *signal error* of layer l as

$$\hat{\delta}^l \equiv \frac{\partial C}{\partial \mathbf{z}^l} = \frac{\partial C}{\partial \mathbf{y}^l}\frac{\partial \mathbf{y}^l}{\partial \mathbf{z}^l} = \frac{\partial C}{\partial \mathbf{y}^l} \circ f_l'(\mathbf{z}^l) \tag{22}$$

where $f_l'(\mathbf{z}^l)$ is the derivative of the activation function of layer l. Clearly then equations (21) may be written as

$$\frac{\partial C}{\partial \hat{W}^l} = \hat{\delta}^l \mathbf{y}^{l-1} \tag{23a}$$

$$\frac{\partial C}{\partial \mathbf{b}^l} = \hat{\delta}^l \tag{23b}$$

where $\mathbf{z}^l = \hat{W}^l\mathbf{y}^{l-1} + \mathbf{b}^l$ was used to evalute $\partial\mathbf{z}^l/\partial\hat{W}^l$ and $\partial\mathbf{z}^l/\partial\mathbf{b}^l$. The next step is to relate the signal error of layer l to $l+1$, the idea to rewrite $\partial C/\partial\mathbf{y}^l$ in (22) using a reverse chain rule:[3]

$$\frac{\partial C}{\partial \mathbf{y}^l} = \frac{\partial C}{\partial \mathbf{z}^{l+1}}\frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{y}^l} = \left[\left(\hat{\delta}^{l+1}\right)^T \hat{W}^{l+1}\right]^T = \left(\hat{W}^{l+1}\right)^T \hat{\delta}^{l+1}$$

That is, the signal error of layer l is related to the signal error of layer $l+1$ via

$$\hat{\delta}^l = \left(\hat{W}^{l+1}\right)^T \hat{\delta}^{l+1} \circ f_l'(\mathbf{z}^l) \tag{24}$$

Equations (23), (22) and (24) are what constitutes the *four equations of the back-propagation algorithm.* The equations essentially boil down to the equivalent of step 3 in Algorithm 1 (in particular (23)). Moreover, note that the "back-propagated quantity" in the BPA is the signal error, i.e., the network's output error is propegated backwards through the network (thereby expressing the dependency of the cost function on the signal error of the different layers).

The back-propagation algorithm is composed of 3 steps: the feed forward cycle, the back-propagation and the network update. The feed forward step refers to the computation of the network's output; the input data $\mathbf{x}$ is "fed forward" to the output. The back-propagation step involves the computation of the signal errors and the final network update is the updating of the network's weights and biases using the derivatives of the cost function. The full algorithm is shown below.

[3] The awkward double-transpose notation is used to preserve the correct dimensions of $\partial C/\partial\mathbf{y}^l$.

Algorithm 2 The Back-Propagation Algorithm

```
1: Initiate the network weights and biases.
2: for $i = 1, \dots, N$:
3:     Feed forward input $\mathbf{x}$.
4:     Compute $\hat{\delta}^L$.
5:     for $l = L-1, \dots, 1$:
6:         Compute $\hat{\delta}^l$.
7:     end for
8:     Update the network weights and biases according to
           $\hat{W}^l_{\text{next}} = \hat{W}^l_{\text{prev}} - \gamma_i \frac{\partial C}{\partial \hat{W}^l_{\text{prev}}}$
           $\mathbf{b}^l_{\text{next}} = \mathbf{b}^l_{\text{prev}} - \gamma_i \frac{\partial C}{\partial \mathbf{b}^l_{\text{prev}}}$
9: end for
```

D. Machine Learning - Support Vector Machines

One of the most versatile and diverse approaches to SML is Support Vector Machines (SVM). SVMs are very popular for both regression and classification problems, this project however will only perform a binary classification. The mathematics are therefore presented with a binary labelling in mind.[4]

1. *The basic principles behind SVMs*

The basic idea of an SVM classifiers is to divide a data set into discrete classes using a *decision boundary*, whose mathematical description is that of a *hyperplane*. Technically speaking, hyperplanes are affine subspaces that represent the direct generalisation of the concept of two-dimensional planes located in three-dimensional spaces. It happens that any decision boundary of a p-dimensional predictor space must be a $(p-1)$-dimensional hyperplane.

Hyperplanes may be parametrised via a linear equation using an intercept β_0 and a parameter vector $\hat{\beta}^T = \begin{pmatrix}\beta_1 & \cdots & \beta_p\end{pmatrix}$. Written in set notation, the definition of a hyperplane in a real p-dimensional space is

$$\text{hyperplane} = \{\mathbf{x} \mid \mathbf{x}^T\hat{\beta} + \beta_0 = 0\} \tag{25}$$

[4] SVM classification with several classes is actually just a recursive use of binary classifications. E.g. Dividing the real number line into three classes $(-\infty, 0)$, $[0, 10]$ and $(10, \infty)$ is the same as first dividing the real number line into $(-\infty, 0)$ and $[0, \infty)$, then subdividing $[0, \infty)$ into $[0, 10]$ and $(10, \infty)$.

This definition of a hyperplane is particularly useful, because it accounts for the two regions separated by the hyperplane by changing the equality in $\mathbf{x}^T\hat{\beta} + \beta_0$ to either $<$ or $>$. Hence, a natural binary classification based on (25) is to simply use the sign:[5]

$$\text{class label} = \text{sign}(\mathbf{x}^T\hat{\beta} + \beta_0) \tag{26}$$

where the binary label (i.e. ± 1) is an arbitrary choice that simplifies notation. Note that the description of decision boundaries as hyperplanes necessarily restricts the possible boundaries to linear boundaries. It turns out that this problem is easily circumvented by introducing a benefitial transformation of the predictor space. This is done in later sections.

Suppose a particular linear decision boundary is parametrised by β_0 and $\hat{\beta}$. By definition, any point $\mathbf{x}_i$ on the hyperplane satisfies $\hat{\beta}^T\mathbf{x}_i = -\beta_0$, and consequently any two points on the hyperplane $\mathbf{x}_1$ and $\mathbf{x}_2$ must therefore satisfy $\hat{\beta}^T(\mathbf{x}_2 - \mathbf{x}_1) = 0$. It follows that $\hat{\beta}/||\hat{\beta}||$ is a unit vector normal to the surface of the hyperplane. Now consider any point in the predictor space $\mathbf{x}$: because the hyperplane is infinite in reach, there must exist some point on the hyperplane $\mathbf{x}_0$ such that the distance from $\mathbf{x}$ to $\mathbf{x}_0$ is minimal. Because $\hat{\beta}/||\hat{\beta}||$ is necessarily the unit vector pointing along the line defined by $\mathbf{x}$ and $\mathbf{x}_0$,[6] the signed distance from $\mathbf{x}_0$ to $\mathbf{x}$ must satisfy

$$\frac{\hat{\beta}^T}{||\hat{\beta}||}(\mathbf{x} - \mathbf{x}_0) = \frac{1}{||\hat{\beta}||}(\hat{\beta}^T\mathbf{x} + \beta_0) \tag{27}$$

That is, the signed distance from any data point $\mathbf{x}$ to $\mathbf{x}_0$ on the decision boundary is proportional to $\hat{\beta}^T\mathbf{x} + \beta_0$. The smallest distance between any data point and the decision boundary is called the decision boundary's *margin* M. The mission statement of SVMs is to find a hyperplane (i.e. determine $\hat{\beta}$ and β_0) that maximises the margin. Using the class labelling

$$y_i = \begin{cases} +1, & \mathbf{x}_i^T\hat{\beta} + \beta_0 \geq 0 \\ -1, & \mathbf{x}_i^T\hat{\beta} + \beta_0 < 0 \end{cases} \tag{28}$$

the SVM mission statement may be expressed as an optimization problem:

$$\frac{y_i}{||\hat{\beta}||}(\mathbf{x}_i^T\hat{\beta} + \beta_0) \geq M, \quad i \in \{1, \ldots, N\} \tag{29}$$

or alternatively if $||\hat{\beta}|| = 1/M$:

$$\min_{\hat{\beta},\beta_0} \frac{1}{2}||\hat{\beta}|| \quad \text{subject to} \quad y_i(\mathbf{x}_i^T\hat{\beta} + \beta_0) \geq 1, \quad \forall i \tag{30}$$

This is an example of a convex optimization problem, and is appropriately studied using the method of Lagrange multipliers.

The Lagrangian primal to be minimized is

$$\mathcal{L}(\hat{\beta}, \beta_0, \hat{\lambda}) = \frac{1}{2}||\hat{\beta}||^2 - \sum_{i=1}^{N} \lambda_i\big(y_i(\mathbf{x}_i^T\hat{\beta} + \beta_0) - 1\big) \tag{31}$$

Its derivatives are

$$\frac{\partial \mathcal{L}}{\partial \hat{\beta}} = \hat{\beta} - \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i \tag{32a}$$

$$\frac{\partial \mathcal{L}}{\partial \beta_0} = -\sum_{i=1}^{N} \lambda_i y_i \tag{32b}$$

Setting $\partial\mathcal{L}/\partial\hat{\beta} = 0$ and $\partial\mathcal{L}/\partial\beta_0 = 0$, then inserting the results back into (31) one finds the Wolfe dual (a maximization problem):

$$\mathcal{L}(\hat{\lambda}) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_i y_i y_j \mathbf{x}_i^T \mathbf{x}_j \tag{33a}$$

$$\text{subject to } \lambda_i \geq 0 \quad \text{and} \quad \sum_{i=1}^{N} \lambda_i y_i = 0 \tag{33b}$$

In addition to the constraints of the Wolfe dual, the solution must also satisfy the so-called KKT (Karush-Kuhn-Tucker) conditions:

$$\lambda_i\big(y_i(\mathbf{x}_i^T\hat{\beta} + \beta_0) - 1\big) = 0, \quad \forall i \tag{34}$$

It follows from (34) that either of two events is possible: If $\lambda_i > 0$, then it must be so that $y_i(\mathbf{x}^T\hat{\beta} + \beta_0) = 1$, in which case $\mathbf{x}_i$ is on the boundary. Else, if $y_i(\mathbf{x}^T\hat{\beta} + \beta_0) > 1$, then $\lambda_i = 0$ and $\mathbf{x}_i$ is not on the boundary. Due to the particular role of the $\mathbf{x}_i$ vectors in the Wolfe dual, they are often called the *support vectors* of the hyperplane (hence the naming of SVM). Computationally the Wolfe dual optimization problem is actually much simpler than the Lagrangian primal optimization problem. Efficient algorithms have been developed and are implemented in most standard optimization software.

Having found the Lagrange multipliers λ_i, the decision boundary hyperplane is found via

$$\hat{\beta} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i \quad \text{and} \quad \beta_0 = \frac{1}{y_i} - \hat{\beta}^T\mathbf{x}_i, \ \forall i$$

The final SVM classifier is given by equation (26).

[5] This classification rule actually does not include points on the decision boundary, thus common practice is to let these points belong to the positive sign class by default.

[6] This follows from the fact that the dimensionality of the hyperplane is one less than the dimensionality of the predictor space. That is, the only degree of freedom that separates $\mathbf{x}$ from $\mathbf{x}_0$ is the direction of the unit vector normal to the hyperplane surface, which is $\hat{\beta}/||\hat{\beta}||$.

2. *Soft classifiers*

The classifier described in the previous section is what is known as a *hard* linear classifier, i.e. a linear classifier that does not allow any cross-over between the two classes. Consequently, hard linear classifiers assume that at least one hyperplane that successfully separates the data set does exist, but this is easily not the case for all data sets. For example, any presence of significant noise could easily produce some data points that "fall on the wrong side" of the decision boundary. Because of its shortcomings, most modern SVM implementations does not use a hard classifier, but instead a so-called *soft* classifier. Soft classifiers are similar to hard classifiers in their structure and purpose, however allows some *slack* with regards to the data set.

Define the slack variables $\hat{\xi} = \begin{pmatrix}\xi_1 & \cdots & \xi_N\end{pmatrix}$, where $\xi_i \geq 0$, and modify equation (29) to "allow some slack" on the form:

$$\frac{y_i}{||\hat{\beta}||}(\mathbf{x}_i^T\hat{\beta} + \beta_0) \geq M(1 - \xi_i) \qquad (35)$$

or alternatively if $||\hat{\beta}|| = 1/M$:

$$\min_{\hat{\beta},\beta_0} \frac{1}{2}||\hat{\beta}|| \quad \text{subject to} \quad y_i(\mathbf{x}_i^T\hat{\beta}+\beta_0) \geq 1-\xi_i,\ \forall i \quad (36)$$

The total *violation* of (29) is now given by $\sum_{i=1}^N \xi_i$, hence bounding the sum from above by some constant C allows for ease of control of the total slack. Equation (36) can be re-expressed in the equivalent form:

$$\min_{\hat{\beta},\beta_0} \left\{ \frac{1}{2}||\hat{\beta}||^2 + C\sum_{i=1}^N \xi_i \right\} \qquad (37a)$$

$$\text{subject to } \xi_i \geq 0,\ y_i(\mathbf{x}_i^T\hat{\beta} + \beta_0) \geq 1 - \xi_i,\ \forall i \qquad (37b)$$

The Lagrangian primal to minimize is now:

$$\mathcal{L}(\hat{\beta}, \beta_0, \hat{\lambda}, \hat{\gamma}) = \frac{1}{2}||\hat{\beta}||^2 + C\sum_{i=1}^N \xi_i - \sum_{i=1}^N \gamma_i \xi_i s \qquad (38a)$$

$$- \sum_{i=1}^N \lambda_i \big(y_i(\mathbf{x}_i^T\hat{\beta} + \beta_0) - (1 - \xi_i)\big) \qquad (38b)$$

From here the derivation follows the same outline as the non-slacked problem, the resulting Wolfe dual to maximize is given by (33a) with the constraints

$$\max_{\hat{\lambda}} \mathcal{L} \text{ subject to } 0 \leq \lambda_i \leq C \quad \text{and} \quad \sum_{i=1}^N \lambda_i y_i = 0 \quad (39)$$

The corresponding KKT conditions are

$$\lambda_i\big(y_i(\mathbf{x}_i^T\hat{\beta} + \beta_0) - (1 - \xi_i)\big) = 0 \qquad (40a)$$

$$\gamma_i \xi_i = 0 \qquad (40b)$$

$$y_i(\mathbf{x}_i^T\hat{\beta} + \beta_0) - (1 - \xi_i) \geq 0 \qquad (40c)$$

for all i. Resorting back to the classifier is identical to the non-slacked case.

Table I: Commonly used classes of kernels in SVM classification.

Class of Kernels	$K(\mathbf{x}, \mathbf{y})$
Linear functions	$c_1\mathbf{x}^T\mathbf{y} + c_2$
n^{th} degree polynomials	$(\mathbf{x}^T\mathbf{y} + c_1)^n + c_2$
Radial Gaussians	$e^{-c_1\|\|\mathbf{x}-\mathbf{y}\|\|^2 + c_2}$
Hyperbolic tangents	$\tanh\big(c_1\mathbf{x}^T\mathbf{y} + c_2\big)$

3. *Non-linear classifiers*

So far, only linear classifiers have been discussed. It turns out that a simple "kernel trick" can be exploited to produce non-linear classifiers. Consider the following basis transformation of the predictor space $\mathbf{x}$ to $h(\mathbf{x})$:

$$h(\mathbf{x}) = \begin{pmatrix}h_1(\mathbf{x}) & \cdots & h_H(\mathbf{x})\end{pmatrix}^T \qquad (41)$$

where H is a positive integer (which can be larger than N). The so-called kernel of $h(\mathbf{x})$ is given by

$$K(\mathbf{x}_i, \mathbf{x}_j) = h(\mathbf{x}_i)^T h(\mathbf{x}_j) \qquad (42)$$

Proceding through the linear classification as outlined in previous sections one arrives on the Wolfe dual

$$\mathcal{L}(\hat{\lambda}) = \sum_{i=1}^N \lambda_i - \frac{1}{2}\sum_{i=1}^N\sum_{j=1}^N \lambda_i\lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \qquad (43)$$

Note that (43) is not expressed in terms of the basis expansion $h(\mathbf{x})$, but instead of the kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$. This is implies that a non-linear classification requires only knowledge of the desired kernel, and not the underlying basis transformation. Common kernels in the SVM literature are shown in table I.

To revert back to the original "hyperplane" (which is now possibly a curved surface) one would necessarily require an expression for the basis expansion $h(\mathbf{x})$. However, this can be avoided by introducing the function

$$f(\mathbf{x}) = h(\mathbf{x})^T\hat{\beta} + \beta_0 = \sum_{i=1}^N \lambda_i y_i K(\mathbf{x}, \mathbf{x}_i) + \beta_0 \qquad (44)$$

Hence, instead of (26), the binary classification of new data points is given by

$$\text{class label} = \text{sign}\big(f(\mathbf{x})\big) \qquad (45)$$

III. METHOD

IV. RESULTS

V. DISCUSSION

VI. CONCLUSION

[1] Integrated pulse profile of a pulsar. Original work by Lorimer Kramer (Handbook of Pulsar Astronomy), republished by AstroBaki, Last Modified December 6th 2017. https://casper.berkeley.edu/astrobaki/index.php/Dispersion_measure.

[2] Jim Condon. Essential radio astronomy, chapter 6: Pulsars. Online resource hosted by National Radio Astronomy Observatory (US), Last modifed 2016. https://www.cv.nrao.edu/~sransom/web/Ch6.html.

[3] F.Graham Smith, F.R.S., former Director of the Royal Greenwich Observatory (1976-1981). *Pulsars*. Cambridge University Press, 1977.

Appendix A: Miscellaneous Material