# Machine Learning Concepts

* This document is just for my own reference. Most of it is taken from different sources on the Internet including brilliant.org.

## 1 Basics

### 1.1 Sum of Squared Error

$$RSS = \sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} WX_i)^2$$

$$X_i = \begin{bmatrix} x_{i0} = 1 \\ x_{i1} \\ \vdots \\ x_{ip} \end{bmatrix}, W = [w_0, w_1, \cdots, w_p]$$

Ridge:

$$\frac{1}{2n}RSS + \lambda \sum_{j=0}^{p} w_j^2$$

Lasso:

$$\frac{1}{2n}RSS + \lambda \sum_{j=0}^{p} |w_j|$$

## 2 Support Vector Machines

A **hyperplane** in $n$-dimensional space is of the form:

$$w_1x_1 + ...w_nx_n - b = 0$$

The vector of weights, $\vec{w}$, is always perpendicular to the hyperplane it corresponds to. For every $n$-dimensional space, there are an infinite number of dimensional hyperplanes to divide it in half. Hyperplane classifiers must work with exactly two different classes, since there are only two sides to a division.

### 2.1 Hard Margin SVM

In a **maximum margin classifier** (AKA the perceptron of optimal stability) the hyperplane separates the data properly (so data needs to be *linearly separable*), and it's as far as possible from the closest points from each class.

It's called perceptron because we are doing a similar thing. We classify based on whether $\vec{w}.\vec{x} - b \geq 0$. But here the perceptron is unique; it is the only hyperplane which meets the given criteria (maximum margin).

Maximum margin hyperplanes don't care about points that aren't close to them. This simplifies the model as we need to deal with fewer points, but it can increase variance. The points that affect the hyperplane are known as **support vectors**.

Suppose $\vec{w}.\vec{x} - b = 0$ separates the data (assume also $\vec{w}.\vec{x} - b = \pm 1$ go through the closest points on each class). The margin is then $2/||\vec{w}||$. So maximal margin classification involves maximizing $2/||\vec{w}||$ or minimizing $||\vec{w}||$.

The process of training a maximum margin hyperplane is computationally expensive. With enough points in the training data set, it can become prohibitively costly to use maximum margin classification. However, these classifiers are worth the effort when there aren't too many points. They are likely to make correct classifications, even when there are more predictor variables than data points and are resilient to outliers and noise.

## 2.2 Soft Margin SVM

A **hinge loss function** takes in a hyperplane and a point. It returns an error of zero if the point is on the correct side of the hyperplane and returns a positive value (based on that point's distance from the correct side of the margin) if it's on the wrong side. Hinge loss of an arbitrary point $x_i$ is:

$$max(0, 1 - y_i(\vec{w}.\vec{x} - b)) \qquad \text{where } y_i \text{ is either 1 or } -1 \text{ and gives the class of point } x_i$$

So it penalizes mis-classified point. This results in a **soft margin SVM**.

The error is then defined as

$$\sum_{i=1}^{n} max(0, 1 - y_i(\vec{w}.\vec{x} - b)) + \lambda||\vec{w}||^2$$

So we want to maximize margin but at the same time minimize the number of mistakes (these are opposite of each other because the bigger the margin the more point are misclassified). The term $\lambda$ determines the emphasis we want to put on each of these two factors.

## 2.3 Non-linear decision boundary

Soft margin SVM works well when the data is *almost* linearly separable. When this is not the case we can transform data to make it so. By adding new variables to the set of predictor variables, it is possible to expand on the space which our data points are inside, making them easy to divide. For example suppose our training data is one-dimensional: class $A = \{-1, 0, 1\}$, class $B = \{-2, -3, 2, 3\}$. This data is not separable by a point (1-dimensional hyperplane). But if we add a predictor $|x|$ to each point and make them two-dimensional, then we can find a line to separate the data.

It is expensive to add a bunch of predictor variables to each data point and then make calculations on them. This is were **kernel functions** come into play. It is much less expensive to define a valid kernel function and then calculate a boundary through it.

As it turns out, In SVM's, the maximization problem (maximizing the margin) can be simplified by considering only the dot products between the points. Dot product is a special case of a kernel function. It describes how similar two vector are. We can form new kernels by adding more variables to the vectors. For example

$$\begin{aligned} K(\vec{u}, \vec{v}) &= \langle u_1, u_2, u_1^2 + u_2^2 \rangle . \langle v_1, v_2, v_1^2 + v_2^2 \rangle \\ &= u_1v_1 + u_2v_2 + u_1^2v_1^2 + u_1^2v_2^2 + u_2^2v_1^2 + u_2^2v_2^2 \end{aligned}$$

Some of the most famous kernels are Linear, Polynomial, Laplacian and Radial Basis Function.

### 2.4 Multi-Class SVM

*one-vs-one classification*: Generate a dividing hyperplane for every pair of classes, then take the majority vote. We will have $\binom{c}{2}$ classifiers and each will be made on $\sim n/c$ points.

*one-vs-all(others)*: Select each class individually and find a boundary which divides that class from every other class. Then to determine class of a point, among dividers that classify the point in that class we choose one that has higher confidence. Confidence is defined as follows: First, we train hyperplanes so that members of the class they test for are on the positive side. This means that the classifier should only output a positive value if $\vec{x_i}$ is in the class it is testing for. The confidence of a classification is simply the magnitude of $\vec{w}.\vec{x} - b$. We will have $c$ classifiers and each will be made on $n$ points.

Most SVM's run in $O(n^2)$ (to produce a dividing hyperplane).

### 2.5 Relation with Logistic Regression

SVM is based on geometry, it finds the dividing hyperplane which produces the largest margin. Logistic regression, on the other hand, is based on statistics. It finds the probability function most likely to produce good results and then generates a dividing hyperplane with it. However, SVMs and logistic regression are very similar mathematically as well as geometrically. As a result, the outputs of the two algorithms are generally quite similar.

Logistic regression is generally used when there is a lot of intersection between classes. When there is no perfect divider, or even a decent one, logistic regression becomes quite useful. If there is a decent divider, or the data is linearly separable, SVM works better.

## 3 Kernels

We know that $\vec{u}.\vec{v} = |u|.|v|.cos(\theta)$. So $\vec{u}.\vec{v}$ increases as the vectors become more parallel (in the same direction). So in a sense, dot product provides a measure of similarity between two vectors. Another measure of similarity may be the distance between them. These are all examples of **kernels**.

The simplest definition for a kernel function is that it is a dot product under some expanded feature space [1]. In other words, if $k(\vec{x}, \vec{y})$ is a kernel, there must be some mapping $\phi$ such that

$$k(\vec{x}, \vec{y}) = \phi(\vec{x}).\phi(\vec{y})$$

Kernels are closed under addition, they are symmetric, $k(\vec{x}, \vec{x}) \geq 0$, and they follow the Cauchy-Schwartz inequality.

Some of kernels that are used more often in ML are as follows. They are used for different types of problems.

| | |
|---|---|
| Linear | $K(\vec{u}, \vec{v}) = \vec{u}.\vec{v}$ |
| Polynomial | $K(\vec{u}, \vec{v}) = (\vec{u}.\vec{v} + r)^n$ |
| Laplacian | $K(\vec{u}, \vec{v}) = e^{-\|\|\vec{u}-\vec{v}\|\|}$ |
| Radial Basis Function | $K(\vec{u}, \vec{v}) = e^{-\|\|\vec{u}-\vec{v}\|\|/2\sigma^2}$ |

Radial Basis Function decreases exponentially as the distance between points increases. It will do a better job accounting for large differences in weight or height.

[1] In the context of machine learning, a feature space is the space of all possibilities for the objects we are studying.

# 4 Reinforcement Learning

- Value Function $v^\pi(s) = E(\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi)$ How good is a state
- $Q^\pi(s,a) = E(\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a\pi)$ How good is a state-action pair
- $Q^*(s,a) = max_\pi Q^\pi(s,a)$
- We are happy with an action $a_t$ in state $s_t$ if $Q^\pi(s_t, a_t) - v^\pi(s_t)$ is large (i.e. action $a_t$ is doing better than average).

# 5 Concepts

## 5.1 MAE vs. RMSE

$$\text{MAE} = \frac{1}{n}\sum |e_i|$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum e_i^2}$$

$$\text{MAE} \leq \text{RMSE} \leq \sqrt{n}\ \text{MAE}$$

- RMSE is more sensitive to outliers. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE should be more useful when large errors are particularly undesirable. So if being off by 10 is more than twice as bad as being off by 5, RMSE should be used. If it is almost as bad, MAE should be used.
- RMSE is better than MAE when the error distribution is expected to be Gaussian.

## 5.2 Mean, Variance and Standard Deviation

- $Var(X) = E[(X-\mu)^2]$
- In a normal distribution, 68% of data is within $1\sigma$ of $\mu$, 95% of data is within $2\sigma$ of $\mu$, and 99.7% of data is within $3\sigma$ of $\mu$.
- $\mu \pm \sigma$ are inflection points of the bell curve.
- A low standard deviation indicates that the data points tend to be close to the mean.

## 5.3 Covariance and Correlation

- The Pearson's correlation coefficient $\rho_{X,Y}$ between two random variables $X, Y$ is defined as

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

- The only restriction is that Pearson's $\rho$ assumes a **linear relationship** between the two variables.
- Pearson's correlation relies on means and standard deviations, which means it is only defined for distributions where those statistics are finite, making the coefficient **sensitive to outliers**.
- Relation of Correlation ($\rho$) and Regression Line ($m$ is the slope of regression line):

$$\rho = m\frac{\sigma_x}{\sigma_y}$$

## 5.4 Bias-Variance Tradeoff

- Bias is the difference between model's expected predictions and the true values. Variance refers to the algorithm's sensitivity to specific sets of training data.
- High bias causes underfitting and high variance causes overfitting.
- Ideally we need a model that has low bias and variance, but this seems not possible because reducing one increases the other.

## 5.5 Accuracy, Recall, Precision, F1

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{T}{T + F}$$
$$\text{Recall} = \frac{TP}{TP + FN}$$
$$\text{precision} = \frac{TP}{TP + FP}$$
$$F1 = 2 * \frac{(\text{precision} * \text{recall})}{(\text{precision} + \text{recal})}$$

**Recall**: Ability of a classification model to find all the data points of interest in a dataset.
**Precision**: Ability of a classification model to identify only the relevant data points.
**F1** (Harmonic mean of Precision and Recal): We use the harmonic mean instead of a simple average because it punishes extreme values (if one recal or precision is very high, the other one has to be very low which makes the F1 score very low).

Suppose we want to identify terrorists trying to board flights. In imbalanced classification problems, accuracy is not helpful. A model that marks everyone as not terrorist gives a very high accuracy (but 0 recall and 0 precision). There is a trade-off between recall and precision: A model that marks everyone as terrorist has perfect Recal (=1.0) but very low precision. The $F1$ score gives equal weight to both measures and is a specific example of the general $F\beta$ metric where $\beta$ can be adjusted to give more weight to either Recall or Precision.

### 5.5.1 Visualizing Recall and precision

- Confusion matrix: shows the actual and predicted labels from a classification problem
- Receiver operating characteristic (ROC) curve: plots the true positive rate (TPR) versus the false positive rate (FPR) as a function of the models threshold for classifying a positive
- Area under the curve (AUC): metric to calculate the overall performance of a classification model based on area under the ROC curve

## 5.6 Bayes' Rule

Conditional Probability:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Bayes' Rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

- The opposite of FP is TN (not TP).
- The FP is the accuracy when test is applied only to negative population.
- Because of the above points, in the context of FP/FN/TP/TN, it's easier to solve the problem using conditional probability:

$$\begin{aligned} P(have\_disease|test\_positive) &= \frac{P(have\_disease \cap test\_positive)}{P(test\_positive)} \\ &= \frac{P(TP) * P(positive\_pop)}{P(TP) * P(positive\_pop) + P(FP) * P(negative\_pop)} \end{aligned}$$

## 5.7 Normalization and Standardization

- Normalization re-scales the data so that every data point $z$ will be in $[0, 1]$:

$$z = \frac{x - min(x)}{max(x) - min(x)}$$

```
from sklearn import preprocessing
x = [[1, 2], [3, 4]]
preprocessing.normalize(x, norm='l1', axis=0)

# array([[ 0.25,  0.33333333],
#        [ 0.75,  0.66666667]])
```

Sometimes you might not want to normalize your data (e.g. the data is proportional, or the scale between your data features does matters). Normalization doesn't handle outliers well.

- Standardization re-scales the data so that the mean is mapped to zero, points greater than mean mapped to a positive number and those smaller than the mean are mapped to negative numbers. The bigger thee standard deviation, the closer the points are to 0:

$$z_i = \frac{x_i - \mu}{\sigma}$$

```
from sklearn import preprocessing
y = [[0, 0], [1, 1]]
scaler = preprocessing.StandardScaler()
scaler.fit(y) # μ = [.5, .5], σ = [.5, .5]
scaler.transform([[2, 2]])

# array([[ 3.,  3.]])
```

Standardizing tends to make the training process well behaved, but it discards some information that may be important.

It handles outliers well but the standardized data may have different scales.

Vahid Vaezian (vahid.vaezian [at] gmail.com), November 11, 2019