Introduccion

Data Mining - Machine Learning - Regression techniques.

La mineria de datos es interdisciplinaria. Encontrar patrones en los datos. Dentro de las tareas de El aprendizaje de maquina es...Los algoritmos de se dividen por la siguiente taxonomía. El aprendizaje supervisado es aprender: ....CUando se denomina clasificacion. Sino se denomina regresion.Dentro de La mineria de datos tiene como objetivo encontrar patrones en un conjunto de datos. Su fin ultimo es poder utilizar estos patrones para poder obtener conocimiento nuevo a partir de un conjunto de datos. La mineria de datos es aprender de los datos. Existen dos enfoques para aprender a partir de un conjunto de datos. El aprendizaje supervisado y el no supervisado. En el aprendizaje supervisado tenemos un conjunto de variables predictoras y un conjunto de y una o mas variables dependientes. El proposito es encontrar una relacion entre estos dos conjuntos que nos permitan predecir cual sera el valor de la variables dependientes a partir de las variables predictoras. Cuando la variable a predecir es del tipo numerica, la tarea de aprendizaje se denomina regresion, cuando esta variable es nominal la tarea se denomina clasificacion.El aprendizaje de maquina, en ingles Machine Learning, es una rama de la inteligencia artificial especializada en desarrollar algoritmos que permitan a la maquina aprender a partir de un conjunto de datos. Dentro de esta area se han desarrollado numerosos algoritmos, tanto en el grupo aprendizaje supervisado como no supervisado. Dentro del primer grupo tenemos algoritmos de regresion como se clasificacion. Los algoritmos de regresion, buscan obtener una funcion f(x) que establesca la relacion

El aprendizaje supervisado(aprendizaje dirigido) consiste en aprender un caracteristica especifica a partir de un conjunto de caracteristicas explicatorias.

Capitulo 2 - Estado del arte

1. Aprendizaje supervisado

Las técnicas de aprendizaje de maquina pueden dividirse en tres grupos. Aprendizaje supervisado, no supervisado, por retroalimentación.

El aprendizaje supervisado consiste en aprender el mapeo entre un grupo de variables predictivas y una o más variables dependientes. Podemos ver este proceso de aprendizaje como dos fases: 1) aprendizaje, 2) utilización de lo aprendido. En la fase 1 se presentan tuplas(X, Y) conformadas por las variables predictivas(X) junto con la variable dependiente (Y). En esta fase cada técnica ajustará su modelo a las tuplas presentadas, dando como resultado un modelo (F). Este modelo (F) será utilizado en la siguiente fase para predecir el valor de la variable dependiente a partir de una tupla conformada únicamente por las variables predictivas. Las técnicas de aprendizaje supervisado pueden subdividirse en dos clases: clasificación, si la variable dependiente es discreta, y regresión cuando la variable dependiente es un valor continuo.

2. Métodos de regresión:

Regresión Lineal

El análisis de regresión implica una variable de respuesta Y y una sola variable predictora X. Esta es la forma de regresión mas simple que podemos encontrar, donde se modela Y como una función lineal de X. Esto es,

*y* = *b*+*wx*; (6.48)

donde la varianza de y es supuesta constante, b y w son los coeficientes de regresion que especifican la intercepción con el eje Y y la pendiente de la recta respectivamente. Los coeficientes de regresion, w y b pueden ser vistos como pesos, haciendo equivalente la siguiente expresión,

*y* = *w*0+*w*1*x*: (6.49)

Estos coeficientes pueden ser resueltos por el metodo de cuadrados minimos, los cuales estiman la recta de mejor ajuste como la que minimiza el error entre el dato verdadero y la estimación de la línea.

Sea |D| el conjunto de entrenamiento que contiene puntos de datos de la forma (x1, y1), (x2, y2), ... , (x|D|, y|D|). Luego los coeficientes de regresión, pueden ser estimados usando este metodo con las siguientes ecuaciones:

Donde es el promedio de x1, x2, ..., x|D|, y el correspondiente a y1, y2, ..., |D|.

Ejemplo 1

Regresión lineal usando el método de mínimos cuadrados. La tabla 1 muestra los pares de datos: años de experiencia laboral de un graduado universitario(X) y el salario correspondiente(Y).

|  |  |
| --- | --- |
| años de experiencia (X) | salario(Y)(miles) |
| 3  8  9  13  3  6  11  21  1  16 | 30  57  64  72  36  43  59ª  90  20  83 |

Tabla 1

Figura 1

El grafico de la figura 1 sugiere una relación lineal entre las dos variables X e Y. De esta manera podemos modelar la relación entre el salario y la cantidad de años de experiencia mediante la ecuación de regresión lineal.

y = w0+w1x

.Utilizando los datos de la tabla 1 junto con las ecuaciones (1) y (2) obtenemos el valor de los coeficientes.

w0 = 23,2

w1 = 3,5:

Luego la recta que mejor se ajusta a los datos de la tabla 1 estará dada por: Y = 23,6 + 3,5X. Utilizando esta ecuación podemos predecir el valor de salario para un nivel de experiencia de la cual no tenemos información. Por ejemplo podemos decir que se estima que un egresado universitario con 10 años de experiencia posee un salario de (= 23,6 + 3,5X \*10 )\*1000= $58600

La regression lineal multiple es una extension de la regression lineal simple de manera de poder incorporar mas de una variable predictora. La misma permite modelar la variable de respuesta como una función lienal de N variables predictoras o atributos *A*1, *A*2, : : : , *An*, formando una tupla, (That is, ***X*** = (*x*1, *x*2, : : : , *xn*).) Nuestro conjunto de entrenamiento D contiene datos de la forma (***X*1**, *y*1), (***X*2**, *y*2), : : : , (***X***j***D***j, *y*j*D*j), donde las Xi son las tuplas de entrenamiento N dimensionales, con etiquetas de clases asociadas Yi. Un ejemplo de un modelo de regresion multiple basado en dos variables predicotas A1 y A2 es: Y = w0 + *w*1*x*1+*w*2*x*2, (6.52)

Donde X1 y X2 son los valores de los atributos A1 y A2 respectivamente de X. El método de cuadrados minimos es extendido para resolver w0, w1, y w2. Sin embargo las ecuaciones son tediosas para resolver a mano. Los problemas de regresion multiple son comúnmente usando paquetes estadísticos.

La regresión lineal es un método simple pero poderoso para ser utilizado en la predicción numérica, el mismo ha sido usado ampliamente en aplicaciones estadísticas durante décadas. La desventaja que presenta este método es la linealidad. Cuando los datos exhiben una dependencia no lineal, la mejor recta de ajuste será encontrada, mediante el método de mínimos cuadrados. Esta recta no se ajustará demasiado bien a este tipo de datos. A pesar de esto los modelos lineales son interesantes ya que sirven como base para el desarrollo de métodos de aprendizaje más complejos.

b. Redes neurales para regresión

Una Red Neuronal Artificial (Red Neuronal o ANN), es un modelo computacional inspirado en las redes neuronales biológicas. La red está conformada de elementos de procesamiento (neuronas), conexiones entre los elementos de procesamiento y coeficientes (pesos) asociados a cada conexión. Todos estos elementos forman la estructura neuronal. Las redes neuronales, en su mayoría, tienen la particularidad de ser sistemas adaptativos, es decir que adaptan su comportamiento de acuerdo al ambiente en donde se encuentran. Así la red es capaz de modificar su estructura, en la fase de aprendizaje, de acuerdo a la información que se le presenta a la misma. En términos prácticos las redes neuronales son técnicas de modelamiento no lineales capaz de modelar funciones complejas. Estas pueden ser aplicadas a problemas de predicción, clasificación o control en un amplio espectro de campos como finanzas, neurociencia, medicina, ingeniería y física.

El bloque fundamental para la construccion de la red neuronal artificial es el modelo matematico de una neurona, como se muestra en la figura. Los tres componentes básicos de una neurona artificial son:

1. Las conexiones que proveen pesos Wj, a los valores de entrada Xj
2. Un sumador el cual que suma las entradas con sus respectivos pesos para computar el valor de entrada a la función de activación.
3. Una función de activación g que mapea v a g(v) el valor de salida de la neurona.



Mientras existen numerosas arquitecturas de redes neuronales, las aplicaciones con mejores resultados en data mining han sido las redes multilayer feedfoward. Estas son redes en las cuales existe una capa de nodos que simplemente aceptan los valores de entrada y capaz sucesivas que son neuronas como las de la figura 1. Las salidas de las neuronas en una capa son entrada a las neuronas en la capa sucesiva. La ultima capa es denominada capa de salida. Las capas entre la entrada y la salida son denominadas capas ocultas, ya que no interactúan con el medio externo. La figura 2 muestra un diagrama para esta arquitectura



En un contexto de aprendizaje supervisado para predicción numérica existe una sola neurona en la capa de salida cuya salida representa la predicción.

*Aprendizaje:*

Una de las características más importantes y que más interesante hace a las redes neuronales es su habilidad para aprender. Si consideramos una red neuronal como una función de mapeo F: X→Y, siendo X un vector de entrada a la red e Y un vector de salida de la misma. Dada una tarea específica para resolver y una clase de función F, aprender significa usar un conjunto de observaciones para encontrar un f\* Ȇ F, que resuelva la tarea específica de manera óptima.

Esta definición implica definir una función de costo C: F→R tal que para la solución optima f\*, C (f\*) <= C (f) para todo f Ȇ F. Es decir que ninguna otra solución tiene un costo menor al costo de la solución óptima. Los algoritmos de aprendizaje realizan una búsqueda en todo el espacio de soluciones para encontrar la solución que menor costo produce.

A pesar que la función de costo puede ser elegida de manera arbitraria, la elección de la misma suele ser realizada basada en las propiedades de la misma (convexidad) y también en las particularidades del problema que se intenta resolver. Finalmente la elección de la función de costo dependerá del tipo de tarea que intentemos resolver. La función de costo mas utilizada en la practica resulta ser la función de minimos cuadrados. F = Sum(Yp – Yv)2

*Algoritmos de aprendizaje:*

Entrenar una red neuronal significa encontrar una función f\* Ȇ F, siendo F: X→Y, tal que minimice el criterio de costo utilizado. El problema de entrenamiento puede dividirse en dos: aprender la estructura de la red y aprender los pesos de las conexiones. Existen numerosos algoritmos de entrenamiento, que resuelven de manera simple los valores de los pesos, dada una estructura de red fija. Ejemplo de ellos son: BackPropagation, Quick Propagation, [Conjugate Gradient Descent](http://en.wikipedia.org/wiki/Conjugate_gradient_method). Por otro lado si bien existen algoritmos para encontrar una estructura de red adecuada este aspecto del problema suele ser resuelto a través de la experimentación.

Los algoritmos de estimación de pesos son, en su mayoría, aplicaciones de la teoría de optimización y de estimación estadística. Estos utilizan alguna variante de la técnica de optimización [Gradient Descent](http://en.wikipedia.org/wiki/Gradient_descent). El algoritmo más conocido es denominado Back Propagation.

La técnica de Back Propagation se compone de un ciclo de dos fases: una fase hacia adelante, donde un dato de entrenamiento se introduce en la red y se calculan las salidas de todos los nodos hasta llegar al nodo final que produce el resultado de predicción. Una fase de retroceso en la cual se van actualizando los pesos de las conexiones desde de los nodos de la capa de salida hasta la capa de entrada.

Backpropagation utiliza Gradient Descent, esta técnica de optimización iterativa usa la información de la derivada de primer orden de la función de costo para ajustar los pesos de la red. A partir del valor de las derivadas, las multiplica por una pequeña constante llamada *tasa de aprendizaje* y luego sustrae el resultado al valor actual del peso. Esto es repetido en cada ciclo del hasta que el cambio en el valor del peso se torna muy pequeño, de esta manera hemos encontrado la configuración de los pesos que logran un minimo de la función de costo elegida.

La taza de aprendizaje determina el incremento en dirección al mínimo y por lo tanto que tan rápido la búsqueda converge. Si esta taza es muy grande y la función tiene múltiples mínimos, la búsqueda puede pasar por alto algún mínimo, o puede oscilar fuertemente. Si la taza es pequeña el progreso hacia un mínimo puede volverse demasiado lento. Cabe destacar que el método de Gradient Descent solo puede encontrar un mínimo local. Si la función de costo tiene varios mínimos puede ser que no se encuentre el mínimo óptimo. Para aliviar este problema suelen realizarse múltiples corridas inicializando los valores de pesos en forma aleatoria.

Como en cualquier otra técnica de aprendizaje de maquina en las redes neuronales podemos sufrir el problema de overfitting, es decir que la red puede reflejar una buena perfomance con los datos de entrenamiento, pero no asi con datos nunca vistos.

*Early stopping* es una modificación a la técnica de gradiente descent la cual resulta en tener un conjunto de datos separado para verificar la perfomance de la red en cada iteración del ciclo de backpropagation. Cuando la performance medida con este conjunto de datos empieza a decaer, indicando overfitting, el algoritmo es terminado. Otro método, denominado *weight decay*, agrega a la función de error una penalización que consiste en la suma al cuadrado de todos los pesos de la red. Esto intenta limitar la influencia de conexiones irrelevantes en las predicciones de la red penalizando los pesos grandes que no contribuyen correspondientemente a una reducción grande del error.

Una pasada por todos los datos de entrenamiento se denomina una Epoch. La mayoría de las aplicaciones de las redes feedforward y de backpropagation requieren varias épocas antes de que los errores sean razonablemente pequeños. El *momentum* es una solución para minimizar el numero de épocas necesarias para encontrar un minimo aceptable. La misma consiste en agregar al peso que se esta actualizando una proporción del incremento agregado en la iteración previa. Esto genera que el proceso de búsqueda sea mas suave(smooth) haciendo los cambios en dirección menos abruptos y favoreciendo una convergencia mas rapida.Valores altos en el parámetro del momentum forzaran a que los ajustes sucesivos sean en direcciones similares. Otra idea es variar el parámetro de taza de aprendizaje para que este comience con un valor alto e ir decrementandolo a medida que se avanza de época. Esto ayuda a bajar el riesgo de overfitting, ya que en épocas tempranas el riesgo de overfitting es bajo y a medida que se incrementa de

Another idea is to vary the adjustment parameter *δ* so that it decreases as

the number of epochs increases. Intuitively this is useful because it avoids

overfitting that is more likely to occur at later epochs than earlier ones.

La técnica de gradient descent resulta ser una de las mas sencillas para aprender los pesos de la red, pero no por esto resulta ser la mas eficiente. En la practica resulta ser bastante lenta. El *momentum* es una solución para mejorar la perfomance de esta técnica. Cuando se esta por actualizar el valor de un peso, se le agrega una proporción del incremento agregado en la iteración previa. Esto genera que el proceso de búsqueda sea mas suave(smooth) haciendo los cambios en dirección menos abruptos.

*Aplicaciones:*

Las redes neuronales pueden verse como una especie de sistema de procesamiento no lineal capaz de resolver un amplio espectro de problemas. Las redes neuronales son útiles cuando existen datos en abundancia pero se carece de una base teórica completa, es decir, no hay un modelo causal o una representación matemática. Los datos disponibles suelen ser no lineales, no estacionarios, o caóticos haciéndolos difíciles de modelar. Las redes neuronales no suponen ningún conocimiento previo acerca del espacio del problema, tampoco necesitan conocimientos previos en cuanto a la distribución estadística de los datos.

Las tareas en la cuales las redes neuronales son aplicadas se encuentran dentro de las siguientes categorías:

[Aproximación de funciones](http://en.wikipedia.org/wiki/Function_approximation), o análisis de regresión, incluyendo predicción de series de tiempo y modelamiento.

[Clasificación](http://en.wikipedia.org/wiki/Statistical_classification), incluyendo reconocimiento de patrones y reconocimiento de secuencias

Procesamiento de datos, incluyendo filtrado, clustering y compresión

Las áreas de aplicación de las redes neuronales incluyen: Sistemas de control(control de vehículos), juegos(backgammon, ajedres), reconocimiento de patrones(sistemas de radares, identificación de caras, reconocimiento de objetos), reconocimiento de secuencias(gestos, habla, escritura), diagnósticos médicos, aplicaciones financieras, descubrimiento de conocimiento en bases de datos, visualización y filtrado de email spam.

c. Árboles de regresión

d. SVM para regresión

Las maquinas de soporte vectorial(SVM) son un conjunto de métodos relacionados al aprendizaje supervisado usados para clasificación y regresión. SVMs pertenecen  a la familia de clasificadores lineales.

Una propiedad especial es que simultaneamente minimizan el error de predicción y maximizan the geometric margin; por esto son conocidos como: maximum [margin classifiers](http://en.wikipedia.org/wiki/Margin_classifier).

*MVS no lineal*

En la mayoría de los casos los datos no son linealmente separables, son problemas no lineales, la ventaja de MVS es que estos problemas pueden ser resueltos mediante la utilizacion de funciones de nucleo(kernel) las cuales permiten hallar la solución sin la necesidad de aplicar explicitamente algoritmos no lineales. Existen dos casos principales para la solución de datos que no son linealmente separables en el espacio de entradas; el primero, cuando los datos pueden ser linealmente separables pero en un espacio de características y, el segundo, cuando no es posible separar dichos datos linealmente incluso en un espacio de características. Ambos casos se explicarán en las subsecciones siguientes.

*MVS lineal en el espacio de características:*

Existen casos en los cuales los datos, debido a su naturaleza, no pueden ser ajustados mediante una funcioin lineal. Sin embargo, en muchas situaciones los datos, a través de una transformación no lineal del espacio de entradas, pueden ser separados linealmente pero en un espacio de características, en el cual se pueden aplicar los mismos razonamientos que para las MVS lineales.

Aunque la dimensión del espacio de características puede ser bastante grande, para ciertas transformaciones y ciertos espacios de características, se puede calcular el producto escalar usando las denominadas funciones núcleo.

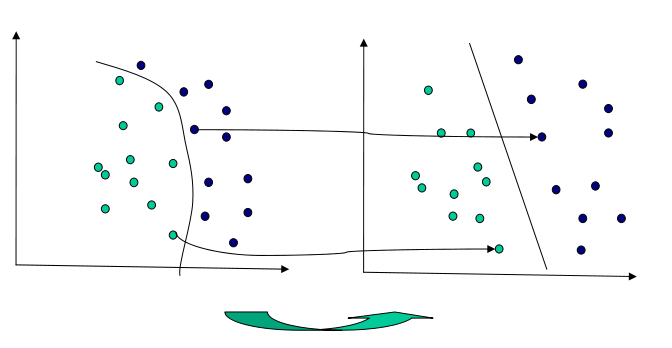
En la figura 5.3 se puede observar un ejemplo donde se puede encontrar un híper-plano óptimo en el espacio de características, a través de un kernel. 

Figura 5.3: Clasificación de MVS para el caso no lineal

Una función núcleo o kernel se puede definir como aquella que permite realizar la separación y el traslado de los datos al espacio de características. Existen diversos kernels predeterminados conocidos entre los cuales se destaca el lineal, el RBF (Función de Base Radial), el polinomial, el sigmoidal, entre otros más. Matemáticamente, un kernel se puede definir como una función K : X × X → R tal que K ( x, y) = < Φ( x), Φ( y) > , donde Φ es una transformación de X en un cierto espacio de Hilbert I . Algunas funciones de kernel son:

Polynomial (homogeneous): 

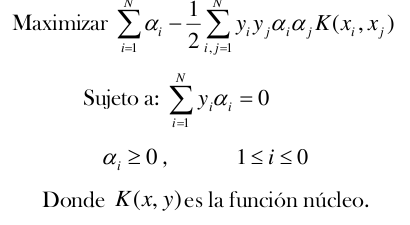
Polynomial (inhomogeneous): 

Radial Basis Function: , for γ > 0

Gaussian Radial basis function: 

[Sigmoid](http://en.wikipedia.org/wiki/Sigmoid_function): , for some (not every) κ > 0 and c < 0

Carreras, Márquez y Romero (2004) afirman que la función núcleo permite calcular el producto escalar < Φ( x), Φ( y) > en el espacio de características sin necesidad de usar ni conocer la transformación Φ . Luego de conocer la idea de lo qué es una función núcleo y qué se puede lograr con ella, se puede proceder a especificar el problema de optimización a resolver para las MVS con margen máximo en el espacio de características. El problema de programación cuadrática con restricciones a resolver es el siguiente:



*MVS con margen blando*

Las MVS con margen blando se refieren a aquellos casos en los cuales los datos no son linealmente separables, ni siquiera en el espacio de características. También es necesario este enfoque cuando los datos pueden ser clasificados linealmente en el espacio de características pero las soluciones son sobreajustadas a los ejemplos y, con esto, se tiende a tener mala generalización. Este modelo es el que precisamente funciona bien con la inclusión de ruido pues resulta ser mucho más robusto que los anteriores modelos explicados.  
 Las MVS con margen blando incorporan variables de holgura, ξi,  al modelo permitiéndole a las máquinas seleccionar un clasificador con cierto margen de error pero que a su vez tienda a tener una mejor generalización. En otras palabras, estas variables, cuyo valor será siempre positivo o igual a cero, permitirán que con cierto margen no se cumplan las restricciones estrictamente. Además, será necesario incluir tantas variables de holgura como datos se tengan, pues para cada instancia de datos se tendrá su respectiva variable de holgura.

La inclusión de variables de holgura a las restricciones en el modelo debe equilibrarse incluyendo en la función objetivo un término de regularización que depende de dichas variables. Este término es el que se conoce como el parámetro C, el cual determina la holgura del margen blando; es decir, el parámetro C es el que permite al clasificador un cierto margen de error en el momento de clasificación. Con esto, el problema a optimizar resulta muy similar al de los modelos con margen máximo, con la diferencia de que en este modelo de margen blando se incluye un término en la función objetivo dependiente del parámetro C y las variables de holgura y, a su vez, a cada restricción se le añadirá las respectivas variables de holgura. Además, se incluirá una nueva restricción que es la que limita el valor del parámetro C.

  
  
Donde ξi >= 0. El híper-plano de separacion optima queda determinado por el vector w que minimiza la funcion:.donde C es un valor dado sujeto a las restricciones de la ecuacion 2.

*Maquinas de Soporte Vectorial para regression :*

El concepto de híper-plano de margen máximo solo puede ser aplicado para clasificación. Sin embargo, se han desarrollado maquinas de soporte vectorial para predicción numérica que comparten muchas de las propiedades descriptas en el caso de clasificación: estas producen un modelo que puede ser expresado en término de unos pocos vectores y puede ser utilizado para problemas no lineales usando una función de núcleo.

Como con la regresión lineal, la idea básica es encontrar una función que aproxime los puntos de entrenamiento minimizando el error en la predicción. La diferencia crucial es que todas las desviaciones hasta un parámetro ɛ dado son descartadas.

Un parámetro ɛ especificado por el usuario define un tubo alrededor de la función de regresión en los cuales los errores son ignorados: para soporte de vectores lineal el tubo es un cilindro. Si todos los puntos de entrenamiento caben dentro de un tubo de 2ɛ, el algoritmo obtiene una función en el medio del tubo más horizontal que los encierra. En este caso el error percibido es cero. La figura muestra un problema de regresión con un atributo, una clase numérica, y ocho instancias. Es este caso el valor de ɛ fue configurado en 1, siendo el ancho del tubo alrededor de la función de regresión igual a 2. La figura X muestra la salida del proceso de aprendizaje con el valor de ɛ configurado en 2. Como se puede apreciar un tubo más ancho hace posible aprender una función más horizontal.

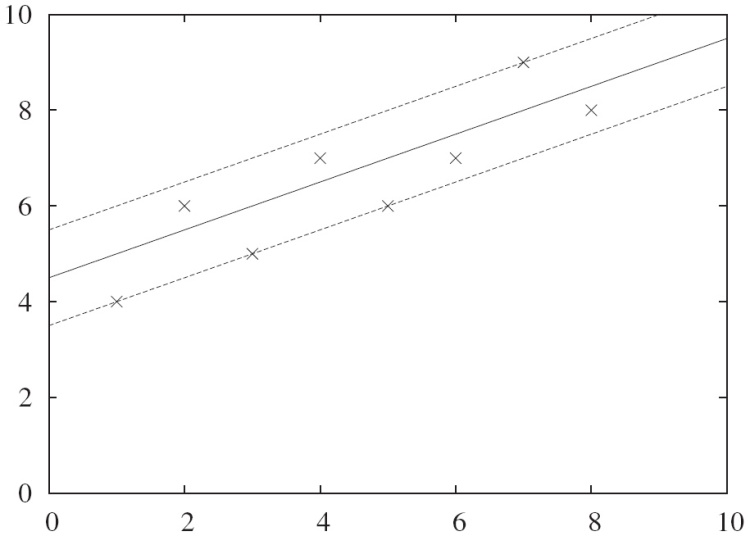
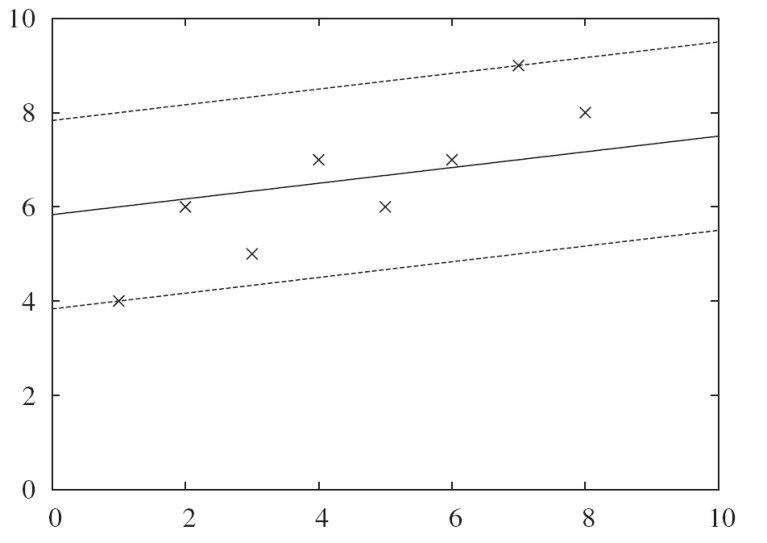
El valor de ɛ controla que tan cerca la función **aproximara(fit?)** los datos. Un valor demasiado grande producirá un predictor sin sentido – en el caso extremo, cuando 2ɛ excede el rango de valores de la clase de los datos de entrenamiento, la línea de regresión es horizontal y el algoritmo solo predice el **valor promedio** (mean) de clase. Por otro lado, para valores pequeños de ɛ puede no haber un tubo que encierre todos los datos. En este caso algunos puntos de entrenamiento tendrán un error diferente de cero, y existirá un “trade-off” entre el error de predicción y la horizontalidad del tubo. En la **figura** **¿?** ɛ fue configurado en 0.5 y no existe ningún tubo de ancho 1 que pueda encerrar todos los datos.

Para el caso lineal, la función de regresión con soporte de vectores puede ser escrita:

Como en el caso de clasificación el producto escalar puede ser remplazado por una función de núcleo para problemas no lineales. Los vectores de soporte son aquellos puntos que no caen estrictamente dentro del tubo –o sea, los puntos afuera del tubo y sobre el borde. Como en la clasificación todos los demás puntos se les asigna coeficiente 0 y pueden ser eliminados de los datos de entrenamiento sin cambiar la salida del proceso de aprendizaje.

El algoritmo busca simultáneamente una minimización del error y una maximización de la horizontalidad de la función de regresión. Sin embargo, cuando algunos puntos no encajan en el tubo**(Figu)** y no existe un tubo con error igual a 0, estamos en presencia de un trade-off entre el error en la predicción y la horizontalidad del tubo. Este “trade-off” es controlado forzando un límite superior C en el valor absoluto de los coeficientes . El límite superior restringe la influencia de los vectores de soporte en la forma de la función de regresión y es un parámetro que el usuario debe especificar en adición a ɛ. Mientras más grande sea C lo más cerca la función **encajara(fit?)** los datos. En el caso degenerado (ɛ=0) el algoritmo simplemente realiza una regresión de *error absoluto minimo* utilizando la restricción del coeficiente y todas las instancias de entrenamiento se transforman en vectores de soporte. Contrariamente si ɛ es suficientemente grande como para que el tubo acomode todos los datos, el error se vuelve 0, no hay “trade-off” para hacer y el algoritmo obtiene el tubo mas horizontal que encierra a los datos indiferentemente del valor de C.

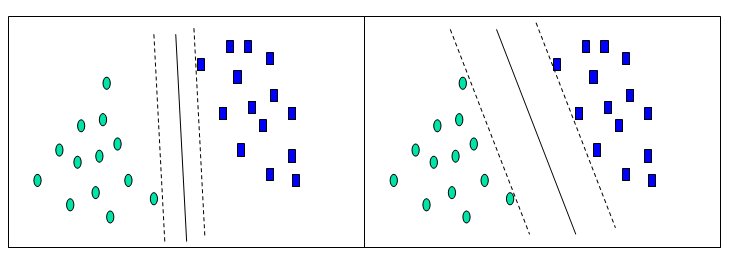




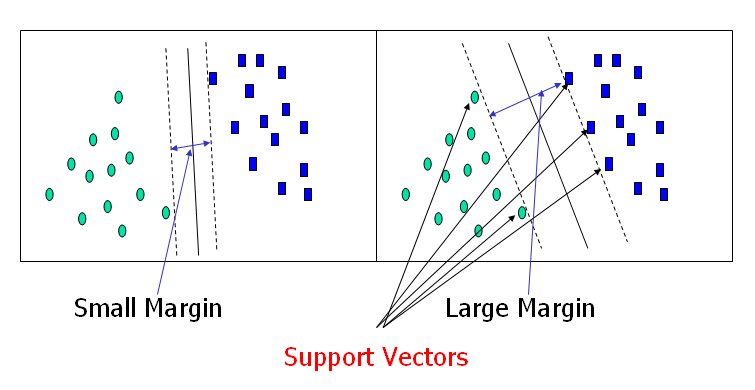
Los datos de entrada se pueden ver como dos conjuntos de vectores en un espacio de n dimensiones, una SVM construye un híper-plano separador en ese espacio que maximizara el margen entre los dos conjuntos de datos. Para calcular dicho margen, se construyen otros dos híper-planos paralelos a este uno a cada lado del original y se van empujando hacia los grupos de datos, en general con márgenes mas grandes se logra mejorar el error del clasificador.  
Los modelos SVM están estrechamente relacionados con las redes neuronales. De hecho, un modelo de SVM que utiliza un sigmoid kernel es equivalente a una red neuronal de dos capas.

Motivation

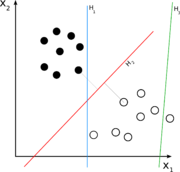
Clasificar datos es una necesidad comun en aprendizaje de maquina. Supongamos algunos datos pertenecientes a dos clases distintas (Espacio de 2-dimensiones), y el objetivo es decidir a que clase pertenece cada dato y lograr separarlos con un hyperplano uni dimensional (una recta). En este caso los vamos a representar con ovalos y rectangulos. Hay un nro infinito de híper-planos posibles pero solo uno es el optimo que maximiza los margenes.



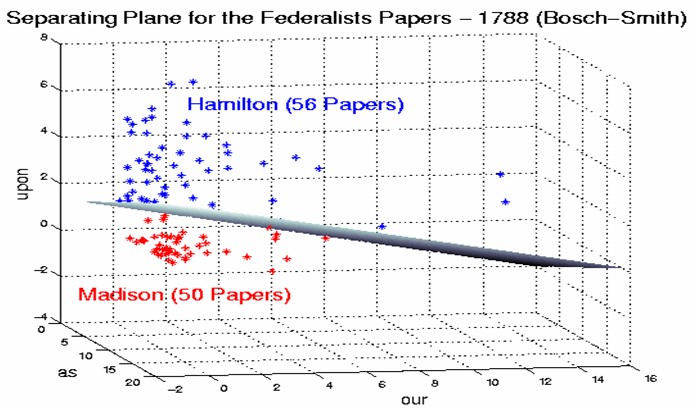
En ambas imágenes, en la figura anterior, vemos un híper-plano de separación posible pero en la segunda vemos el optimo. Las líneas punteadas muestran la distancia entre el híper-plano y los puntos más cercanos a él. Los puntos ubicados al lado de las líneas punteadas que delimitan el margen máximo de dicho híper-plano se denominan vectores de soporte, y se destacan en la figura siguiente:



Estamos interesados adicionalmente en encontrar una máxima separación (margen) entre las dos clases de datos. Es decir que queremos encontrar el híper-plano cuya distancia a un dato sea la mayor. Si encontramos este híper-plano lo llamaremos híper-plano de máximo margen.

  
H3 (verde) no separa las dos clases. H1 (azul) las separa con un margen pequeño y H2 (rojo) lo hace con el máximo margen.

Si todos los análisis consistieran en dividir dos clases distintas de datos, todo resultaría muy sencillo. Por lo general esto no sucede por lo que SVM tiene que lidiar (a) con más de dos variables predictivas, (b) separando los datos con curvas no lineales, (c) manejando casos donde los datos no puedes ser completamente separados y (d) manejando clasificaciones con más de dos categorías.  
  
En el ejemplo previo solo teníamos dos variables predictivas, y pudimos exponer los datos en un plano de dos dimensiones. Si tuviéramos 3 variables predictivas usaríamos un cubo de 3 dimensiones. Datos en un plano de 2 dimensiones pueden separarse con un un híper-plano unidimensional, similarmente datos en un cubo de 3 dimensiones pueden separarse con un híper-plano bidimensional:



A medida que vamos agregando variables predictivas(atributos), los datos pueden ser representados en un N-dimensional space, y un (N-1)dimensional hyperplane pueden separarlos.

En el caso de SVM, un dato es visto como un vector p-dimensional, y si queremos podemos separar esos puntos con un híper-plano de p-1-dimensiones. Esto es llamado clasificación lineal. hay muchos híper-planos que pueden clasificar los datos, sin embargo.

Formalization

Dado algunos datos de entrenamiento, un conjunto de puntos de la forma:



donde ci es tanto 1 or −1, indicando la clase del punto  . Cada  es un p-dimensional [real](http://en.wikipedia.org/wiki/Real_number) vector. Queremos ofrecer el maximum-margin hyperplane que divide los puntos con clase ci = 1 de los ci = − 1. Cualquier híper-plano puede ser escrito como un conjunto de puntos  que satisface:



El vector  es un [normal](http://en.wikipedia.org/wiki/Surface_normal) vector: y perpendicular al híper-plano. El parametro  determina el desplazamiento del híper-plano desde el origen a lo largo del normal vector .

Queremos elegir el  y b para maximizar el margen entre los híper-planos paralelos para que esten los mas separados posible manteniendo los conjuntos de datos separados. Estos híper-planos pueden ser descriptos con las ecuaciones

  
y

Notar que si los datos de entrenamiento son linealmente separables podemos seleccionar los dos híper-planos del margen de manera que no queden datos entre estos y luego intentar maximizar su distancia. Usando geometria, podemos encontrar que distancia entre estos dos híper-planos es , por lo tanto queremos minimizar . Al mismo tiempor tenemso que prevenir que los datos caigan dentro del este margen por lo que agregaremos las siguientes restricciones : para cada  i

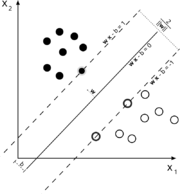
 para  de la primera clase o  
 para  de la segunda.

Esto se puede reescribir asi:



Podemos poner esto juntos para obtener el problema de optimizacion:

elegir  para minimizar sujeto a 





Maximum-margin hyperplane and margins for a SVM trained with samples from two classes. Samples on the margin are called the support vectors.

Primal Form

El problema de optimizacion presentado en la seccion anterior es dificil de resolver porque depende del valor absoluto de |w|. Afortunadamente es posible alterar la ecuacion sustituyendo  ||w|| por  sin cambiar la solucion (the minimum of the original and the modified equation have the same w and b). Esto es un [quadratic programming](http://en.wikipedia.org/wiki/Quadratic_programming) (QP) [optimization](http://en.wikipedia.org/wiki/Optimization_(mathematics)) problem. Mas claramente,

minimizar , sujeto a .

El factor 1/2 es usado por convenienza matematica. This problem can now be solved by standard quadratic programming techniques and programs.

Dual Form

La escritura de la regla de clasificacion en su forma dual sin restricciones revela que el maximum margin hyperplane asi como la tarea de clasificacion es solo una funcion de los support vectors, los datos que se encuentran en el margen. The dual of the SVM can be shown to be:

 subject to , and 

where the α terms constitute a dual representation for the weight vector in terms of the training set:



3. Problema de Predicción de oleaje

El problema de prediccion de oleaje es el siguiente:

4. Trabajos relacionados a la predicción de oleaje (con redes neurales u otros métodos)

Los trabajos relacionados a la prediccion de oleaje:

NAME

weka.classifiers.functions.MultilayerPerceptron

SYNOPSIS

A Classifier that uses backpropagation to classify instances.

This network can be built by hand, created by an algorithm or both. The network can also be monitored and modified during training time. The nodes in this network are all sigmoid (except for when the class is numeric in which case the the output nodes become unthresholded linear units).

OPTIONS

GUI -- Brings up a gui interface. This will allow the pausing and altering of the nueral network during training.

\* To add a node left click (this node will be automatically selected, ensure no other nodes were selected).

\* To select a node left click on it either while no other node is selected or while holding down the control key (this toggles that node as being selected and not selected.

\* To connect a node, first have the start node(s) selected, then click either the end node or on an empty space (this will create a new node that is connected with the selected nodes). The selection status of nodes will stay the same after the connection. (Note these are directed connections, also a connection between two nodes will not be established more than once and certain connections that are deemed to be invalid will not be made).

\* To remove a connection select one of the connected node(s) in the connection and then right click the other node (it does not matter whether the node is the start or end the connection will be removed).

\* To remove a node right click it while no other nodes (including it) are selected. (This will also remove all connections to it)

.\* To deselect a node either left click it while holding down control, or right click on empty space.

\* The raw inputs are provided from the labels on the left.

\* The red nodes are hidden layers.

\* The orange nodes are the output nodes.

\* The labels on the right show the class the output node represents. Note that with a numeric class the output node will automatically be made into an unthresholded linear unit.

Alterations to the neural network can only be done while the network is not running, This also applies to the learning rate and other fields on the control panel.

\* You can accept the network as being finished at any time.

\* The network is automatically paused at the beginning.

\* There is a running indication of what epoch the network is up to and what the (rough) error for that epoch was (or for the validation if that is being used). Note that this error value is based on a network that changes as the value is computed. (also depending on whether the class is normalized will effect the error reported for numeric classes.

\* Once the network is done it will pause again and either wait to be accepted or trained more.

Note that if the gui is not set the network will not require any interaction.

autoBuild -- Adds and connects up hidden layers in the network.

debug -- If set to true, classifier may output additional info to the console.

decay -- This will cause the learning rate to decrease. This will divide the starting learning rate by the epoch number, to determine what the current learning rate should be. This may help to stop the network from diverging from the target output, as well as improve general performance. Note that the decaying learning rate will not be shown in the gui, only the original learning rate. If the learning rate is changed in the gui, this is treated as the starting learning rate.

hiddenLayers -- This defines the hidden layers of the neural network. This is a list of positive whole numbers. 1 for each hidden layer. Comma seperated. To have no hidden layers put a single 0 here. This will only be used if autobuild is set. There are also wildcard values 'a' = (attribs + classes) / 2, 'i' = attribs, 'o' = classes , 't' = attribs + classes.

learningRate -- The amount the weights are updated.

momentum -- Momentum applied to the weights during updating.

nominalToBinaryFilter -- This will preprocess the instances with the filter. This could help improve performance if there are nominal attributes in the data.

normalizeAttributes -- This will normalize the attributes. This could help improve performance of the network. This is not reliant on the class being numeric. This will also normalize nominal attributes as well (after they have been run through the nominal to binary filter if that is in use) so that the nominal values are between -1 and 1

normalizeNumericClass -- This will normalize the class if it's numeric. This could help improve performance of the network, It normalizes the class to be between -1 and 1. Note that this is only internally, the output will be scaled back to the original range.

reset -- This will allow the network to reset with a lower learning rate. If the network diverges from the answer this will automatically reset the network with a lower learning rate and begin training again. This option is only available if the gui is not set. Note that if the network diverges but isn't allowed to reset it will fail the training process and return an error message.

seed -- Seed used to initialise the random number generator.Random numbers are used for setting the initial weights of the connections betweem nodes, and also for shuffling the training data.

trainingTime -- The number of epochs to train through. If the validation set is non-zero then it can terminate the network early

validationSetSize -- The percentage size of the validation set.(The training will continue until it is observed that the error on the validation set has been consistently getting worse, or if the training time is reached).

If This is set to zero no validation set will be used and instead the network will train for the specified number of epochs.

validationThreshold -- Used to terminate validation testing.The value here dictates how many times in a row the validation set error can get worse before training is terminated.