Resumen: Organización de datos para el final

Introducción a Machine Learning

Llamamos **Machine Learning** a la rama de la computación que se encarga de construir algoritmos que aprendan a hacer algo útil a partir de los datos.

Evaluación de los Algoritmos de ML

Los algoritmos de Machine Learning se dividen en dos grandes grupos:

- **Algoritmos de aprendizaje supervisado**: clasificación automática.
  - *Problemas de regresión*: en este tipo de problema queremos predecir el valor de una variable numérica y continua a partir de un cierto conjunto de datos. El objetivo es construir un modelo que nos permita predecir el valor de nuestra variable de decisión a partir de datos nuevos.
  - *Problemas de clasificación*: es muy similar a la regresión pero la variable que queremos predecir no es continua sino discreta, frecuentemente tiene pocos valores posibles y en muchos casos los valores posible son solo dos. La clase es una variable discreta con un set de valores posibles limitado y definido. La idea es la misma, contamos con un set de entrenamiento en el cual para cada dato conocemos la clase a la cual pertenece el mismo, queremos construir un modelo que nos permita clasificar automáticamente datos nuevos cuya clase desconocemos.
  - *Problemas de recomendaciones*: el objetivo de un sistema de recomendaciones es recomendarle al usuario cosas que pueden interesarle. Involucran el esfuerzo conjunto de varios algoritmos y herramientas.
  - *Problemas de Sistemas de consultas*: es un buscador, search engine, el contenido que se almacena puede ser de cualquier formato aunque en general se almacenan en páginas HTML o texto plano (información no estructurada). El objetiva del sistema de consulta es recuperar los textos o páginas o ítems de información más relevantes para la consulta planteada por el usuario.
  - *Problemas de identificación de patrones*: Data mining: el objetivo es descubrir información interesante a partir de un conjunto de datos. En conclusión: contamos con información que frecuentemente es de tipo transaccional y queremos encontrar patrones, asociaciones, entre los ítem que se incluyen dentro de cada transacción.
- **Algoritmos de aprendizaje no supervisado**: problemas de clustering.
  - *Problemas de Clustering:* contamos con los datos que queremos dividir en grupos de forma automática. En algunos casos la cantidad de çlusters"la debemos indicar previamente y en otros el algoritmo es capaz de determinarla por sí mismo. A estos problemas se los suele llamar .ªprendizaje no supervisado"La diferencia está dada porque no necesitamos conocer el valor de una cierta variable o clase para cada punto, es decir que sólo necesitamos los datos en crudo y el algoritmo es capaz de encontrar los clusters automáticamente.

Casi todos los algoritmos de ML se basan en la costrucción de un **modelo** a partir de los datos, de forma tal de luego poder usar dicho modelo para predecir datos nuevos. Cada modelo tiene un conjunto de:

- Parámetros: los descubre el algoritmos a partir de los datos.

- Hiper-parámetros: datos que debemos pasarle al algoritmo para funcionar.

El **problema** que surge es: *¿cómo encontrar los hiper-parámetros óptimos?*

En general se trabaja con un **set de entrenamiento** y un **set de datos**. La idea es que entrenaremos a nuestro algoritmo con el set de entrenamiento y luego lo aplicaremos al set de test.

Para poder validar el modelo necesitamos dividir el set de entrenamiento original en dos partes: un **set de entrenamiento** y un **set de validación**. La idea es entrenar el modelo con el set de entrenamiento y luego probarlo con el set de validación a efectos de encontrar los mejores hiper-parámetros.

Resumen: probamos distintos valores de hiper-parámetros mediante grid-search o random-search y luego elegimos los hiper-parámetros que mejor resultado nos den en el set de validación. Luego aplicamos al set de test el modelo con los hiperpármétros que encontramos.

Importante: es **incorrecto** utilizar el set de test como set de validación, ya que el algoritmo nunca debe considerar los datos de test para encontra los valores óptimos para uss hiper-parámetros.

Problema: bajo este esquema siendo el set de validación siempre el mismo podemos caer en el problema de que los **hiper-parámetros que encontremos sólo sean óptimos para un pequeño conjunto de nuestros datos.** Para evitar esto usamos el método de **cross-validation** el cual es prácticamente universal para otpimizar algoritmos de Machine Learning.

`Grid-search`: es la simple y exhaustiva búsqueda manual de un conjunto de huper-parámetros del espacio de hiper-parámetros posibles. Pruebo todas las combinaciones posibles. Toma mucho tiempo.

`Random-search`: doy valores para cada hiperparámetro, pero en este caso pruebo n combinaciones al azar, es decir, que no prueba todas las combinaciones posibles. Esta solución se aplica para problemas en los que tengo el tiempo limitado.

Cross-Validation

El proceso de K-fold Cross Validation comienza particionando el set de entrenamiento en k bloques. Luego vamos a realizar varias iteraciones en las cuales entrenamos nuestro algoritmo con k-1 bloques y lo validamos con el restante. Este proceso se repite k veces para que todos los datos hayan participado alguna vez del set de vaidación. El resultado es el promedio de las k iteraciones del algoritmo. Esto hahy que hacerlo por cada valor posible para nuestros hiper-parámetros, por lo que, dependiendo de los datos, puede resultar un proceso costoso (un caso extremo es aquel en el que usamos un sólo dato en el set de validación).

Overfitting vs Underfitting

Overfitting: el concepto está asociado a la complejidad del modelo. Un modelo excesivamente complejo puede ajustar tan bien como queremos al set de entrenamiento pero funcionar muy mal para el set de test.

Underfitting: es el opuesto. Se produce cuando el modelo es demasiado simple.

Podemos decir que el **modelo óptimo** es aquel que tiene la complejidad necesaria para capturar lo que los datos expresan pero no más.

Para entender cuándo tenemos UF o OF necesitamos los siguientes conceptos:

- **Bias**: el error que tenemos en el set de entrenamiento. Puede asociarse al poder de representación.
- **Variance**: el error que tenemos en el set de test (o validación). Puede representarse a la variablidad del poder de representación.

A medida que aumenta la complejidad del modelo, el error del set de entrenamiento disminuye, este es el Bias (Bias siempre decreciente)

A medida que aumentamos la complejidad, el error del set de test aumenta, este es el Variance.

El óptimo está en algún lugar en el medio. Por lo tanto:

- Cuando el modelo no es óptimo porque le falta complejidad tenemos alto bias, baja varianza y **underfitting**.
- Cuando el modelo es demasiado complejo tenemos bajo bias, alta varianza y **overfitting**.
- Cuando tenemos alta varianza y alto bias el modelo es un desastre. No entiende los datos y además tiene demasiada variablidad.

Podemos plotear los errores y cuando tengamos un caso de underfitting veremos que las curvas de error nunca se aproximan, la distancia entre el error de test y de entrenamiento es reflejo de un modelo que no tiene la complejidad necesaria para entender los datos.

Cuando el problema es la alta varianza tenemos que las curvas se aproximan pero el error total es demasiado alto. Este es un caso de overfitting, el modelo es demasiado complejo y generaliza mal.

El teorema de No Free Lunch (NFL)

Teorema: dos algoritmos de optimización cualesquiera son equivalente si los promediamos sobre el set de **todos** los problemas posibles.

Corolario: Dado un problema de otimización, si un algoritmo funciona muy bien, entonces existe un problema en el cual el algoritmo funciona igual de mal.

Corolario del corolario: no existe un algoritmo que sea óptimo para cualquier problema de optimización.

Hay una poderosa analogía entre el teorema NFL y el teorema fundamental de la compresión de datos. N**ingún algoritmo de compresión puede comprimir cualquier archivo y ningún algoritmo de optimización puede otpimizar cualquier problema.**

Es posible que UN algotimo funcione mejor para cualquier problema de optimización que tenga sentido, de la misma forma que hau un algoritmo de compresión que suele ser el mejor en general (ej: PAQ). Es fundamental ditinguir el concepto teórico del teorema NFL con el concepto fundamental, necesario y poderoso de que así como los archivos comprimidos no son random, lo problema de ML tampoco lo son. **Los datos no son random.**

Ensambles

Los mejores algoritmos de ML suelen surgir de la **combinación de varios algritmos.** Es muy raro que un sólo algoritmo de ML logre mejores resultados que un **ensamble** (no quiere decir que en la práctica sea conveniente usar ensambles).

Bagging: en general implica aplicar el mismo clasificador n veces y luego promediar sus resultados para obtener el resultado final. El problema es que aplicar el mismo clasificador n veces el mismo resultado. Es por esto que **bagging** siempre se usa en conjunto con **bootstrapping**.

Bootstrapping consiste en tomar una muestra del set de entrenamiento del mismo tamaño del set de entrenamiento pero con reemplazo. Es decir que un dato puede estar varias veces en la muestra.

Entonces, podemos entrenar nuestro clasificador con los bootstraps y obtener n clasificadores distintos. Luego podemos aplicar esto son clasificadores al set de datos original y promediar los reultados para obtener el resultado final. Esto sirve para problemas de regresión como de clasificación.

Como cada clasificador no ve el total de registros del set de entrenamientos la técnica de bagging disminuye notablemente las posiblidades de caer en **overfitting**, ya que ninguno de los n clasificadores individuales puede sobre-ajustar al set de entrenamiento completo.

Para cada clasificador existe un set de registros que queda afuera, a estos los llamamos registros **out of bag** (OOB). La precisión de un clasificador que usa bagging se puede obtener mediante la clasificación de los registros OOB con el clasificador mismo, como si los registros OOB sirvieran como set de validación de cada clasificador individual.

El promedio de presición para los registros OOB se suele usar para analizar la precisión del ensamble entero. De esta forma buscaremos los hiper-parámetros que nos den mejor promedio de precisión OOB. Esto evita que la búsqueda de hiper-parámetros haga overfit.

$$arboles\ de\ decision\ +\ bagging\ =\ randomforrest$$

Boosting: consite en construir un algoritmo muy preciso a partir de un conjunto de algoritmos muy simples, los cuale spor separado pueden funcionar bastante mal. El método consiste en:

- Entrenar un algoritmo simple.
- Analizar sus resultados.
- Entrenar otro algoritmo simple en donde se le da mayor peso a los resultados para los cuales el anterio tuvo peor performance. Cada algoritmo teien a su vez un peso proporcional a la cantidad de aciertos que tuvo para el set de entrenamiento.
- El resultado final del algoritmo se construye mediante un promedio ponderado de todos los algoritmos usados con sus respectivos pesos.

Combinación de diferentes algoritmos de clasificación

Majority Voting

Tenemos varios clasificadores distintos para un cierto problema, cada uno de ellos produce un resultado y queremos obtener un resultado final. Una aproximación simple es ver cual es la clase que tiene mayoría entre todos los clasificadores. Este tipo de ensamble tiene sentido cuando la predicción es directamente la clase. Si la predicción es la probabilidad de cada clase entonces otros

métodos funcionan mejor.

El resultado del voto por mayoría mejora notablemente si se usan reultados que tengan poca correlación entre sí. Entonces **dado muchos clasificadores es conveniente elegir un conjunto que tenga buenos resultados y estén poco correlacionados**. Este será luego el set de clasificadores que usaremos para el ensamble.

Es lógico pensar que el modelo que mejor está funcionando es el que en general tiene mayor cantidad de aciertos y, por lo tanto, solo queremos cambiar sus preddciones si muchos de los demás clasificadores más débiles opinan lo contrario. Esto es simplemente darle un peso a cada modelo. Un esquema simple es darle n votos al mejor clasificador, n-1 votos al siguiente, etc. Luego simplemente aplicamos majority voting, pero el mejor clasificador aparece varias veces en la votación y, por lo tanto, su opinión será más importante.

Averaging

Promediar el resultado de varios clasificadores es un método muy popular que funciona en muchos problemas distintos. La idea principal es reducir el overfitting.

Cuando promediamos clasificadores que predicen la probabilidad de las clases hay que tener cuidado porque cada clasificador individual puede tener una calibración completamente diferente. Para prevweir esto se puede convertir cada probabilidad de un rango entre 1 y n, siendo n el total de registros. El registro con mayor probabilidadd tiene 1 el segundo 2, ... etc hasta n, sin importar el valor de las probabilidades. SI hacemos esto para todos los clasificadores podemos luego promediar los rangos y convertir estos promedios en un número entre 0 y 1 para la probabilidad final. Para calcular la probabilidad final simplemente normalizamos $prob = \frac{x-min}{max.min}$

Blending

Uno de los mejores resultados de la creación de ensambles a partir de clasificadores diferentes. La idea es entrenar varios clasificadores diferentes y armar un set de datos con sus predicciones para luego entrenar otro clasificador que realice las predicciones finales en base a la combinación de otros clasificadores.

Pasos:

- Separar un 10 % del set de entrenamiento (no es el set de validación).
- Entrenar n clasificadores diferentes con el 90 % restante del set de entrenamiento (separar este 90 % en sets de entrenamiento y validación para optimizar cada modelo.
- Realizar n predicciones para el 10 % que separamos usando cada uno de los n clasificadores que entrenamos.
- Entrenar un modelo blender que use las predicciones aprendidas para realizar la clasificación final.
- Entrenar los n modelos con el set de entrenamiento completo.
- Realizar predicciones con estos modelos apra el set de test.
- Combinar las predicciones para el set de test usando el modelo blender.
- Combinar los set de entrenamietno y test en un nuevo súper set de entrenameinto en donde los labels para lo que era el set de test son los que predijo el blender.

- Separar un 10 % sel súper test.
- Entrenar los n modelos en el 90 % restante del súper set.
- Predecir los resultados para el 10 % que estamos separamos para cada modelo.
- Entrenar un blender para combinar estas predicciones en la predicción final.
- Entrenar los n modelos usando el súper-set completo.
- Aplicar los modelos aprendidos al set de test.
- Aplicar el súper blender al set de test.

KNN

KNN (k-Nearest-Neighbors) se basa en encontrar para un determinado punto sus K-vecinos más cercanos. Asumimos que nuestro set de dato está formado por un conjunto de m puntos en n dimensiones siendo todos los valores numéricos.

Para poder usar KNN hay que definir dos cosas:

- La métrica a usar para calcular las distancias.
- El valor de k.

Estos son los hiper-parámetros.

Cuando queremos simplemente clasificar un punto cuya clase no cocnocemos no hace falta las probabilidades podemos simplemente asignarlo a la clase con mayoría entre los k-vecinos del punto.

Si nuestro problema es de regresión entonces podemos predecir para nuestro punto el promedio de los valores de los k-vecinos más cercanos.

Métrica a emplear

La métrica debe cumplir las siguientes propiedades:

- Debe ser positiva.
- Debe ser simétrica.
- Debe cumplir con la desigualdad triangular.

Distancia de Minkowsky

Definición: $\left(\sum_{i}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$

Cuando $\mathbf{p = 0}$ es la distancia de Hamming o Norma l_0 o de Minkowssky. En este caso la distancia entre dos puntos es igual a la cantidad de elementos en las cuales difieren los vectores. La distancia de Hammming en general se define y se usa para vectore binarios, formados por cers y unos pero la definición puede extenderse a vectores con valores arbitraris contando simplemente en cuantas dimensiones los valores de los vectores son diferentes.

Cuando $\mathbf{p = 1}$ es la distancia de Manhattan

Cuando $\mathbf{p = 2}$ es la distancia euclideana.

Cuando $\mathbf{p} = \infty$ es la norma l_∞ y la distancia equivala a la diferencia más grande entre dos elementos cualesquiera ente los vectores.

Distancia de Mahalanobis

Es útil cuanto detemos atributos (features) que están correlacionados. Es aquella distancia que teniendo en cuenta la variabilidad de cada uno de los atributos logra que puntos que están dentro del mismo percentil de variación para su atributo queden a la misma distancia. La formula es:

$$D(x, y) = \sqrt{(x - y)S^{-1}(x - y)}$$

Donde S es la matriz de covarianza que se define como aquella que tiene en la diagonal la varianza de cada uno de los atributos y en los demas elementos la covarianza entre los atributos. Se define la covarianza de la siguiente manera:

$$cov(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mathbf{x})(y_i - \mathbf{y})$$

Donde las letras negritas deberían tener una barrita encima.

Cuando la matriz de covarianza es la identidad la distancia de Mahalanobis es igual a la distancia euclideana.

Distancia coseno

Mide la distancia entre dos vectores como el ángulo entre los mismos.

$$cos(\theta) = \frac{< x, y >}{|x||y|}$$

De ahí hay que despejar el ángulo theta.

Notamos que el coseno es una medida de semejanza (a mayor cosieno más chico el ángulo entre los vectores) mientras que el ángulo es una medida de distancia.

La distancia coseno se usa cuando lo que importa es la dirección en la que apuntan los vectores y no la matnitud de los mismos.

La distancia coseno está realacionada con el coeficiente de correlación de Pearson (falta la fórmula de pearson)
La distancia Coseno o el coeficiente de Pearson son especialmente útiles cuando nuestros vectores tienen muchos elementos deconocidos, o lo que es igual, son muy dispersos, con mayoría de ceros. En estos casos si calculamos la distancia euclideana sólo para los elementos del vector que son conocidos los vectores que tienen muy pocos elementos (o muy pocos distintos de cero) van a estar muy cerca de todos los demás. Es por esto que el coseno o Pearson son usados en sistemas de recomendaciones.

Distancia de Edición o Levenshtein

Forma de calcular distancias entre strings. Lo que medimos es la cantidad de operaciones que tenemos que hacer para convertir un string en otro. Las operaciones válidas son agregar o quitar

un caracter del string, ambas tienen costo uno y la distancia es la suma de los costos.

La particularidad de esta distancia es que es **costoso** calcularla.

Distancia de Jaccard

Se usa para calcular la distancia entre conjuntos.

Definición: $dJ(X,Y) = 1 - \frac{X \cap Y}{X \cup Y}$

La semejanza de Jaccard es 1 menos la semejanza de jaccard y es un número entre 0 y 1.

Distancia Geodésica

Se aplica en grafos y es simplemente la cantidad de aristas a recorrer para llegar desde un nodo a otro. Esta distancia se usa cuando no tenemos información sobre las coordenadas de los nodos sino simplemente la información de qué nodos están vinculados con otros.

Distancia entre grafos

Esta distancia sirve para comparar un grafo con otro, es decir, queremos saber que tan diferentes son dos grafos cualesquiera. Es similar a la distancia de edición entre strings, contamos con el costo de las operaciones necesarias para convertir un grafo en otro. Las operaciones poibles son agregar un nodo nuevo, eleminar un nodo, agregar una harista o eliminar una arista, todas ellas de costo 1.

Distancia para atributos categóricos: VDM

Definición $VDM(f_1, f_2) = \sum_{class=i}^{c} |P(f_1|class_i) - P(f_2|class_i)|$ f1, y f2 son los valores del atributo categórico que puede tomar.

Distancias especiales

Es para definir una distancia mixta o sutomizada. A cada coincidencia de cada valor categórico le asigna un puntaje. La semejanza máxima que puede haber entre dos puntos es el máximo de puntos que podemos sumar cuando todos los atributos son identicos. Luego la semejanza entre dos puntos se puede calcular como la suma de puntos sobre el total posible y la distancia es 1 - semejanza ya que la semejanza es un número entre 0 y 1.

Determinar distancias

El método básico es probr diferentes distancias y ver cual de ellas nos da un mejor resultado. Como conocemos la clase del set de entrenaminto lo que hacemos es comparar las predicciones con los valores conocidos para calcular la presiición del algoritmo o el error del mismo. En un problema de clasificación podemos calcular el porcentaje de puntos que el algoritmo clasifica correctamente mientras que en un problema de regresión podemos calcular la sumatoria total de los errores al cuadrado sobre el total de puntos (MSE).

Este mecanismo nos permite analizar el comportamientod e KNN en el set de entrenamiento y nos ayuda a elegir mejores hiper-parámetros para el algoritmo. Al usar este método de validación cuando tomamos cada punto no lo consideramos con parte de sus k.vecinos, es decir que hacemos

de cuenta que el punto está fuera del set de entrenamiento. Si los puntos fueran muchos podemos seleccionar algunos al azar para calcular la precisión del algoritmo.

Determinar el valor de k

porbar diferentes valores de k y ver cuál es el que nos da mejores reulstados. Es conveniente usar valores de k grandes, sin embargo, al aumentar el valor de k estamos dando cada vez mayor peso a las clases que tienen una mayor cantidad de puntos en el set de entrenamiento. En definitiva el k óptimo en KNN es aquel que nos da un buen desempeño en cuanto a la precisión de clasificación para el mayor k posible.

Sensibilidad fuera de escala

Un punto débil de KNN es que para que funcione correctamente todos los atributos de nuestros datos deben tener el mismo peso en el cálculo de distancias, si los atributos o estan todos dentro de la misma escala entonces puede que un atributo domine el cálculo de distancias sobre todos los demás y el algoritmo tendra un rendimiento pobre. Para evitar que un atributo domine el cálculo de distancias es necesario normalizar los valores de los atributos, es decir restándole a cada atributo el promedio de la cada columna y dividiendo por el desio standar de la misma. Este proceso se llama Normalización.

Sensibilidad a atributos anómalos

Es posible que no todos los atributos sean adecuados para clasificar. En muchos casos esto no es evidente ya que desconocemos cuáles son los atributos más relevantes y cuáles son no relevantes. Métodos para determinar atributos relevantes:

- **Foward Selection**: comenzamos con cero atributos y en cada paso agregamos el atributo que mejor resultado nos genera. Para considerar el resultado podemos usar un valor fijo de k o bien la mejor presición para varios valores de k o bien un promedio de la precisión para difrentes valores de k probados. De esta forma vamos iterando y agregando atritbutos siempre y cuando los resultados mejoren.
- **Backward Selection**: es la versión inversa de FS. Comenzamos con todos los features y vamos eliminando un feature a la vez hasta que no se pueda mejorar la precisión del algoritmo.

Aproximaciones para KNN

La eficiencia dle algoritmo en su versión más simeple para clasificiar un punto nuevo tenemos que compararlo contra todos los puntos existentes m para enconta los k vecinos más cercanos. Cada una de estas m comparaciones implica comparar n dimensiones. Por lo tanto el orden del algoritmo es O(m*n).

La necesidad de comparar cada punto a clasificar contra todos los puntos del set de datos es el gran problema de KNN

KNN no escala bien a partir de una cierta cantidad de datos es necesario usar algún tipo de optimización o aproximción que nos permita lograr mejores resultados.

Aproximaciones posibles:

- Indices espaciales: KD-trees:

  Un ídice espacial es una estructura de datos en donde podemos insertar nuestro puntos n-dimensionales de forma tal de luego poder realizar búsquedas de los puntos más cercanos a un cierto query sin tener que recorrer todos los puntos del set de datos.

  Es similar a un árbol binario de búsqueda pero en cada nivel del árbol comparamos una coordenada diferente de nuestro puntos n-dimensionales. Buscamos que las ramas estén balanceadas para que el algoritmo sea de orden logaritmico y no lineal (la mediana funciona mejor que el promedio).

  El problema de KD-trees y de todos los índices espaciales en general es que sólo funcionan bien para muy pocas dimensiones: 2,3 o 4. A medida que aumenta las dimensiones los índices se degradan muy rápidamente $->$ MALDICIÓN DE LA DIMENSIONALIDAD. Esta última dice que algunos algoritmos funcionan bien con pocas o muchas dimensiones, en este caso, con pocas. Es muy raro contar con un problema de Data Science en el cual los datos se presentan en dos o tres dimensiones.

- Indices Espaciales: VP-trees (Vantage Point Tree):

  También sufren la maldición de la dimensionalidad pero resisten mucho más a la misma que un KD-Tree. Vantage Point Tree es una estructura de datos que no necestia ningún perámetro pero si que la distancia a utilizar cumpla con la desigualdad triangular.

  Empezamos considerando a todos los puntos en un único nodo. Elegimos al azar un punto que será nuestro vantage point inicial. Calculamos la distancia desde dicho punto a todos los demás y calculamos la mediana de todas las distancias. Luego dividimos los puntos en dos conjuntos: aquellos cuya distancia es menor o oigual a la mediana y aquelllos cuya distancia es mayor a la media. La raiz del árbol contiene el Vantage point, la mediana de las distancias que llamaremos μ y dos punteres uno al sub-árbol izquierdo con todos los puntos a distancia menor igual a la mediana y otro al sub-arbol derechos con todos los puntos con distancia mayor a la media.

  Cuando queremos realizar una query empezamos por la raíz. Si la distancia desde el punto query q a la raiz (τ) es menor a μ entonces calculamos si con un radio de τ estamos siempre dentro del radio de μ alrededor del punto raíz. Si esto ocurre sólo tenemos que explorar la rama de izquierda del árbol usando la distancia entre el query y la raiz como valor para τ.
  Si con un radio de μ al rededor del punto query quedamos siempre fuera del círculo de radio μ entonces tenemos que explorar el súbarbol derecho usando $\tau = \mu$.
  En el peor caso estamos en un caso intermedio entre los dos anteriores y tenemos que explorar ambos sub-árboles. Una vez que accdemos al sub árbol izquierdo o derecho analimaos la distancia entre el punto query y la raiz y repetimos el proceso.

  Este índice espacial no es perfecto y cuando trabajamos con muchas dimensiones puede darse que en todos los casos haya que recorrer los dos hijos de cada nodo lo cual degrada en una buúsqueda lineal o peor. Esto ocurre rápidamente en datos sintéticos y aleatorios, incluso con pocas dimensiones pero como los datos reales nunca son random podemos suponer que no se distribuyen de forma uniforme por lo que un VP-Tree puede darnos una ventaja cuando la cantidad de puntos con la que estamos trabjando es realmente muy grande incluso en muchas dimensiones.

Esto nos muestra que existe unapuja constante entre la maldición de la dimensionalidad y la bendición de la no-uniformidad de los datos y difrentes algoritmos que explotan esta tensión de forma diferente.

- Lideres y seguidores

  Es un algoritmo aleatorizado. Necesita una tetapa de pre-procesamiento de los datos que se hace una única vez y luego de esta etama permite aproximar los vecinos más cercanos sin necesidad de comprar contra tdos los puntos.

  Algoritmo: en primer lugar tomamos una cantidad de puntos al azar de nuestro set de datos, en generar $\sqrt{n}$ y los denominamos líderes. Luego procesamos cada uno de los puntos que quedaron del set de datos y comparando contra cada líder lo asignamos al líder más cercano. De esta forma luego de la etapa de pre-procesamiento cada punto del set de entrenamiento o bien es un líder o está asociado (linkeado) a un líder). Por último para buscar los k vecinos más cercanos a un cierto query lo que hacemos es buscar comparar el punto contra cada líder y para el líder más cercano comparamos contra todos sus seguidores. Es decir que hacemos únicamente comparaciones con $2\sqrt{n}$ puntos ya que suponemos que cada lídert tiene asociados en promedio $\sqrt{n}$ seguidores.

  Es posbible mejorar la precisión del algoritmo sacrificando la velocidad cuando comparamos contra los n líderes más cercanos en lugar de uno sólo.

- Aproximación con K-means:

  Similar a líderes y seguidores. Lo que hace es aplicar K-means a los datos tomando $\sqrt{m}$ centroides (m es la cantidad de puntos). K-means es un algoritmo de clustering que nos devuelve un conjunto de centroides y a cada punto lo asigna al centroide más cerrcano.

  Cuando queremos los k vecionos más cercanos a un punto comparamos contra los centroides y los puntos que pertenecen a ese cluster. Eventualmente podemos comparar contra los b centroides más cercanos si queremos mejor precisión con el costo de algunas comparaciones más.

  Este método difiere al anterior en el sentido que los líderes no son elegidos al azar sino son los centroides encontrados por K-means lo que sugiere que estan mejor distribuidos en el espacio. La desventaja es que tenemos que correr k-means sobre el set de datos lo cual es menos eficiente que tomar los líderes al azar.

- Editing(ESTE NO LO ENTENDÍ):

  Consite en eliminar del set de entrenamiento puntos que no son necesarios para la clasificación de otros puntos. Consideramos que para un cierto valor fijo de k nuestro espacio de puntos definidos por el set de entrenamiento queda dividido en áres que corresponden a cada una de las clases posibles, cualquier punto futuro que caiga dentro de cada área será clasificado con la clase que corresponde al área dentro de la cual ha caido el punto.

  Notemos que para definir cada área sólo es necesario contar con los puntos que están cerca de las fornteras de las mismas, los puntos interioores no juegan ningún papel en la clasificación de puntos cuya clase desconocemos. El proceso de editing tiene como objetivo eliminar dichos puntos. Los puntos que son relativamente relevantes para clasificar los vamos a llamar prototipos.

Para encontrar los porototipos hay dos procedimientos backward selection y forward colection (COMPLETAR ACÁ).

El proceso de editing mejora la performance de KNN y no cambia en absoluto la precisión del algoritmo para clasificar ya que solo remueve de nuestro set de entrenamiento aquellos puntos que no son necesarios para clasificar. El costo del algoritmo es la ncesidad de preprocesar todos los puntos para obtener los prototipos (esto se hace 1 vez).

- NN vía grafos:

  Lo que hacemos es unpreprocesamiento de los datos para construir un grafo donde los nodos son los puntos y los vamos a linkear a sus k vecinos más cercano, es decir que, el grafo tiene tantos nodos como puntos y cada punto tiene k aristas.

  Para buscar los k puntos más cercanos a un cierto query lo que hacemos es comenzar desde un punto al azar del grafo y calcular la disntancia entre el query y los vecinos de dicho punto. Una vez que hicimo esto nos movemos al nodo que nos acerca al query. En el camino nos queamo con los k puntos que haya estado más cerca del query que estamos procesando.

  Este algoritmo es bastante rápido y eficiente siempre y cuando sea factible realizar el pre.procesamiento necesario.

- LSH:

  LSH algoritmos para determinar rápidamente datos que son similares entre sí. La idea es usar funciones de hashing especiales que asigne a una misma posicsión datos que son similares. Pueden usarse para aproximar los algoritmos de KNN con lo cual ahora se llamarán ANN (aproximated nearest Neighbours). Por cada punto aplicamos la función de hashing y luego comparamos contra todos los puntos que estén dentro del bucket. **Esto permite aproximar los k vecinos más cercanos a un cierto punto en O(1).**

Teoría de KNN

Vamos a desarrollar el error que comete KNN al clasificar. Para ello definimos el error mínimo que puede tener un clasificador que llamaremos e*. El error siempre lo vamos a calcular con un número de 0 y 1 que es la probabilidad de que clasifiquemos mal un punto. si todos los puntos son iguales lo mejor que podemos hacer es predecir la clase mayoritaria para todos los puntos y nuestro error será igual a la proporción de puntos que no están en esta clase.

Analizaremos ahora cuál es el error cuando k = 1, es decir, NN (Nearest Neighbor).
Teorema de Cover-hart: Si e* es el error óptimo de un clasificador entonces si tenemos infinitos datos el error de NN es eNN $\leq$ 2e*.

Esto nos asegura que NN tiene en el peor de los casos el doble de error que un clasificador ideal. Una conclusión interesante de la desigualdad es que si el error óptimo es e* y el error de KNN con k=1 es a lo sumo 2e* entonces cuando tomamos k ¿1 nuestro mejor resultado es a lo sumo el doble de tomar k = 1. Esta es una cota muy improtante porque nos permite probar métircas con k =1 y luego analizar cuánto pueden mejorar tomando mayor cantidad de vecinos.

La desigualdad de Cover-hart puede extenderse a KNN y lo que obtenemos es que para KNN el error es alo sumo el error ideal multiplicado por una cosntante c muy chica, esto quiere decir que

si los datos fueran infinitos KNN sería ótpimo. Hay que tener cuidado porque **los datos nunca son infinitos por más grande que sea el set**.

Conclusión final: KNN mejora a medida que tenemos mayor cantidad de puntos, cuanto mejor definidas queden las fronteras mejor será la performance del algoritmo en clasificar puntos nuevos.

Parzen windows

Es un algoritmo muy parecido a KNN. En lugar de seleccionar siempre a los k vecinos más cercanos lo que hace es seleccionar a los vecinos que etan dentro de una cierta distancia prefijada, por lo tanto, la cantidad de vecinos es variable.

En cuanto a los resultados obtenidos en general ambos algoritmos dan resultados bastantes similares salvo que Parzen Windows suele usarse para problemas de regresión.

KNN con pesos

La idea es darle pesos a los vecinos de acuerdo a su proximidad con respecto al punto que queremos estimar. Esto es lógico en problemas de regresión en donde podemos razonar que si nuestro punto está realmente muy cerca de otro entonces el valor a estimar debería ser muy parecido al de dicho punto sin que vecinos más alejados influyan mucho. El uso de pesos invalida el proceso de editing ya que ahora todos los puntos del set de entrenamiento pueden ser útiles.

La idea es asignarle a cada uno de los k vecinos un peso:

$$W_(i \neq 1) = \frac{d(x, x_k) - d(x, x_i)}{d(x, x_k) - .d(x, x_1)}$$

El punto más cercano siempre tiene peso 1 y el más lejano tiene peso 0.

Una vez que calculamos los pesos es sencillo estimar el valor de regresión para el punto en cuestión como:

$$Y = \frac{\sum_i w_i + x_i}{\sum W_i}$$

Notemos que también podemos usar este esquema para problemas de clasificación. En KNN antes cada vecino tenía un voto para determinar a qué clase pertenece el punto, ahora tendrá el valor correspondiente a W_i y la clase que sume más es la que determina la del nuevo punto.

Otra forma popular de pesar los vecinos de KNN es mediante la fórmula:

$$W_i = \frac{1}{d(x, x_i)^\beta}$$

Donde β es un parámetro. Cuando $\beta = 0$ es el algoritmo tradicional donde todos los k tienen peso uno. Cuando $\beta = 1$ tenemos una interpoción lineal, etc.

Por ultimo una tercera opción es pesar los puntos mediante un Gaussiano al rededer de cada punto:

$$w_i = exp\left(\frac{|x_i - x|^2}{2\sigma^2}\right)$$

en donde σ es el radio al rededor de cada punto para considerar que los puntos dentro d elos mismo son vecinos. Podemos determinar σ calculando el promedio de las distancias entre cada punto y

su k-esimo vecino más cercano.

Con los pesos podemos usar un valor de k tan grande como deseemos, los puntos más lejanos simplemente van a tener un peso cada vez menor y en concreto podríamos usar k= n en cuyo caso para clasificar un punto tenemos en cuenta todos los puntos del set de datos como vecinos. Cuando esto pasa lo que tenemos es efectivamente un kernel de distancias entre puntos.

Evitando el Overfitting en KNN

Cuando k es chico tenemos riesgo de Overfitting. Para evitar el eerror de generalización podemos eleminar del entrenamiento puntos **anómalos definiendo como punto anómalo aquel para el cual todos sus vecinos pertenecen a una clase diferente a la del punto**. Luego de eliminar los putnos anómalos la capacidad de genralizar debe ser superior ya que se elimina el efecto de estos puntos en la frontera del algoritmo.

RKNN: ensambles basados en KNN

KNN es muy sensible a los atributos que usamos para calucular las distancias podemos entonces crear un ensamble en donda cada KNN usará un conjunto de m atributos al azar de nuestro set de datos.
La cantidad de m atributos es un hiper-parámetro al igual que k y la métrica a usar como distancia, los tres hiperparámetros tiene que buscarse por grid-search usando cross-validation.

El uso de un esnamble permite minimizar el impacto de atributos que no son buenos predictores o que afiectan el resultado de KNN, un detalle muy importante es que el ensamble nos permite calcular la importancia predictora de cada atributo. Esto lo podremos usar para detectar cuáles son los atributos que ayudan más a KNN y cuáles son los menos significativos y eliminarlos, realizando un nuevo ensamble con menor cantidad de atributos posibles. Este es un caso de feature-selection basado en un ensamble de KNN's.

Mainfolds

Es un espacio que localmente se comporta como un espacio euclideano aunque globalmente no lo sea.

En muchos casos un conjunto de datos se presenta en un espacio que no corresponde a la dimensionalidad real de los datos, cuanto tenemosun espacio representado en otro hablaremos de un embedding", este puede ser mayor o menor cantidad de dimendiones que el espacio original. A los algoritmos que intentan descubrir la verdadera dimendionalidad de los datos lo llamaresmo algoritmos de Manifold Learning.

Los datos no son random y siempre tiene pocas dimensiones

La premisa que intentaremos explicar es que independientemente de la cantidad de dimensiones en las cuales se presenten los datos estos casi siempre tienen pocas dimensiones. Es decir que frecuentemente nuestro datos se van a presentar como un manifold de pocas dimensiones inmerso en un espacio dimensional mucho mayor.

Para explicar esto es clave entender que los datos no son aleatorios, si fueran aleatorios serían ruido. Que los datos tengan un sentido implica que existe una estructura en los mismos y esto

implica que no sean aleatorios. Como los datos no son aleatorios es imposible que cubran todo el espacio dimensional en el cual se presentan.

Manifold Learning y cambios de dimensiones

Los algoritmos de manifold learning y cambio de dimensiones permiten tranformar los datos de un espacio dimensional a otro, para hacer esto tiene que existir algún motivo válido. Motivos comunes:

- Para poder visualizar los datos en dos o tres dimensiones.
- Porque el algoritmos que queremos usar funciona mejor en otra dimensión (combatir la maldisión de la dimensionalidad).
- Por razones de eficiencia de tiempo o espacio.
- Para eliminar el ruido de nuestro set de datos.

Maldición de la dimensionalidad

Definición: No todos los algoritmos se comportan bien en cualquier espacio de dimensiones. Algunos algoritmos se comoprtan bien en pocas dimensiones y otros se comportan mejor en muchas dimensiones. Es **importante** entender que para cada algoritmo hay razones completamente diferentes por las culaes podemos preferir muchas o pocas dimensiones.

El problema del muestreo: en todo problema de Data Science es correcto decir que cuantos más datos podamos recolectar mejor serán nuestros resultados. En 2001 Banko y Brill publicaron un trabajo que decía que con la cantidad de datos suficientes incluso algoritmos muy simples convergen al mismo resultado que los algoritmos más avanzados. Aún así tenemos que analaizar *el efecto de la dimensionalidad* en la cantidad de datos necesarios. sabemos que a medida que aumenta la cantidad de dimensiones del set de datos aumenta exponencialmente la cantidad de datos "posibles"por lo tanto a mayor cantidad de dimensiones necesitamos exponencialmente más datos para mantener el mismo nivel de muestreo (sampling).

Podemos decir que esto es un efecto de la maldición de la dimensionalidad sin embargo a pesar que a mayor cantidad de dimensiones la cantidad de datos posibles aumenta exponencialmente si y sólo si cualquier dato es posible pero esto implicaría datos aleatorios y hemos establecido que l**os datos nunca son aleatorios**. Por lo tanto,hay que tener cuidado al decir que a mayor cantidad de dimensiones necesitamos muchos más datos para tener un buen muestreo, en la mayoría de los casos con datos reales esta afirmación no es válida.

El efecto de la dimensionalidad sobre las distancias: analizaremos este efecto para la distancia general de Minowsky. Existe un teorema que nos dice que a medida que la cantidad de dimensiones tiende a infinito la diferencia entre la distancia máxima entre dos puntos del espacio y la distancia mínima converge es decir que todas las distancias son aproximadamente iguales. Sin embargo, vimos que los datos no suelen ocupar completamente el espacio en el cual se presentan pero de todas formas es importante considerar que e**n muchas dimensiones el concepto de distancia entre putntos empieza a peligrar**.

Por lo tanto a medida que aumenta la cantidad de dimensiones exponenetes cada vez más bajos nos dan una mejor medida de distancia entre los puntos. Se demuestar que para muchas dimensiones la distancia de Mahattan (p = 1) funciona mejor que la distancia euclideana e incluso con $p < 1$ funcionan aún mejor.

Shared Neighbors: Ademas del uso de exponentes fraccionarios podemos usar el concepto de vecinos más cercanos compartidos o shared nearest neighbors. Esta métrica define la distancia entre dos puntos x e y como la intersección entre los vecinos más cercanos de ambos puntos.

$$SNN(x, y) = |KNN(x, k) \cap KNN(y, k)|$$

Esta métrica funciona mejor que eotras en espacios de muchas dimensiones.