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 estos clasificadores al set de datos original y promediar los resultados 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 a lo 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.

Clasificación

Siempre contaremos con un set de entrenamiento que tiene n registros en d dimensiones y con cada registro tenemos asociado un **label** que nos dice a qué clase pertenece el registro. El objetivo de un algoritmo de clasificación es construir un modelo que permita predecir la clase de datos que no pertenecen al set de entrenamiento, es decir, datos nuevo.

Cuando hay dos clases hablamos de clasificación binaria y cuando tenemos 3 o más hablamos de clasificación multiclase.

Árboles de decisión

Es un árbol binario en donde en cada nodo dividimos el set de datos en dos de acuerdo a un cierto criterio. El **objetivo** es llegar a nodos hoja en los cuales podamos clasificar correctamente los datos.

ID3 (iterative dichotomister tree)

Es un algoritmo de tipo greedy, en cada paso realiza el mejor split posible del set de datos en dos partes. Este proceso se repite recursivamente hasta construir el árbol final. Lo que necesitamos para construir el árbol es determinar qué split realizar en cada paso y en ID3 el criterio es: *seleccionar el atributo que nos da mayor ganancia de información.*

La ganancia de información está relacionada directamente con el concepto de entropía.

La **entropía del set de datos** se calcula como la entropía en el que las probablidades son la P(label) y la P($\bar{label}$).

Para calcular la ganancia de información de un atributo debemos calcular la entropía de cada valor posible. Para ello:

atributo	label	$\bar{label}$
val 1	cant 1	$\bar{cant}$ 1
val 2	cant 2	$\bar{cant}$ 2

P(val 1 — cant) = $\frac{cant}{cant+\bar{cant}}$

Luego las probablidadades las usamos para calcular la entropía de cada valor del atributo.

La entropía del atributo es el promedio ponderado de la entropía de cada valor posible por la probabilidad de que el atributo tome dicho valor. Es decir la probabilidad de que el atributo tome dicho valor es la cantidad de casos (label) sobre la cantidad de casos posibles (label + $\bar{label}$)

La **ganancia de información** es la diferencia entre la entropía del set de datos y la entropía del atributo.

Utilizamos para clasificar el atributo de mayor ganancia de información. Cuando un valor de este atributo no pude decidir es decir que tiene igual probabilidad de ser clasificado de una u otra manera lo que hacemos es elegir el proximo atributo con mayor ganancia de información y agregar otro nodo a nuestro árbol de decisión.

Un árbol de decisión corre un grave peligro de caer en **overfitting**, en definitiva, podemos hacer pregutnas y preguntas hasta que cada hoja tenga únicamente nodos de una sóla clase. Para evitar esto suele usarse un hiper-parámetro minbucket que indique la cantidad mínima de registros que puede tener un nodo no-hoja.

C4.5

Es una mejora con respecto a ID3, ellas son:

- Aceptar atributos numéricos, para manejarlos se establece un umbrar para realizar el split. Cuando sucede esto lo que hacemos es ordenar sus valores y luego buscar linealmente cuál es el split que nos da menor entropia ya que a menor entropía mayor ganancia de información.
- Aceptar datos con atributos faltantes. Se ignoran en el calculo de la entropía.

- Permitir que los atributos tenga un cierto peso en cuyo caso se pondera la ganancia de información por el peso del atributo para decidir los splits.
- Podar el árbol una vez credo. El algoritmo examina el árbol e intenta remover las ramas que no ayudan en nada reemplazándolas con nodos hoja, esto ocurre cuando la predicción que haríamos en un nodo es igual o mejor a la predicción a la que llegamos en sus hojas.

Random forest

Es uno de los algoritmos más populares en clasificación porque suele producir buenos resultados para la mayoría de los set de datos, el gran secreto es su enorme habilidad para evitar overfitting y la fuerza de funcionar en base a un **ensamble**.

En líneas generales un RF es una aplicación directa de bagging a árboles de decisión pero con una diferencia, cada árbol no usa el total de atributos sino un subset de los mismos.
Los RF tienen, en general, dos hiper parámetros:

- La cantidad de árboles a crear: En general mayor cantidad de árboles mejor funciona el ensamble. Hay que tener en cuenta que llegado un punto la mejora es prácticamente inapreciable y perdemos performance.
- La cantidad de atributos de cada uno. Necesita ser encontrado mediante grid-search usando la precisión OOB.

El hecho de usar un bootstrap en lugar del set de entrenamietno completo permite evitar overfitting mientras que el usar un subconjunto de atributos tomados al azar ayuda a determinar cuáles son los atributos que realmente resultan más importantes para la clasificación.

Un RF es capaz de darnos una medida de importancia para cada atributo promediando la precisión OOB de aquellos árboles que usan el atributo.

Con cientos o miles de árboles un RF puede tener una muy buena opinión sobre la capacidad predictora de cada atributo por lo que podemos usarlos como parte del proceso de feature engineering para decidir qué atributos eleiminar del set de datos por considerarlos ruidosos.

Distancia RF

Un RF nos permite aprender una métrica para nuestro set de datos. Si queremos calcular la distancia entre x_i y x_j lo que hacemos es clasificar a ambos puntos con nuestro RF aprendido en base al total de los puntos y luego tomar como distancia el número de árboles en los cuales la predicción para x_i y x_j es diferente. Dividiendo por el total de árboles tenemos una distancia normalizada entre 0 y 1.

Xgboost

Este algoritmo es **el estado del arte** de los algoritmos de clasificación. Lo que busca (como en todos los algoritmos de clasificación) es disminuir el error (ajustar bien los datos) y diminuir la complejidad del modelo (evitar el overfitting).

En **Boosting** la predicción se contruye mediante la suma de resultados de varios árboles. La clave de boosting es que cada árbol intentará corregir los errores del anterior.

Naive bayes

Está basado en el teorema de Bayes. Los pasos son:

- Construir una matriz con la cantidad de veces que aparececada palabra en cada clase (porque estamos clasificando palabras de un tipo de documento).
- Lo que queremos calcular es:

$$P(class|D) = \frac{P(class)P(D|class)}{P(D)}$$

- P(class): es la cantidad de documentos de dicha clase sobre el total de documentos.
- P(D): la podemos suponer 1, en realidad no nos interesa porque vamos a comparar la probabilidad del documento en cada clase y el denominador es siempre el mismo así que podemos ignorarlo.
- La probabilidad de un documento dada una clase depende de las palabras que aparezcan en el documento y la probabilidad de estas palabras en dicha clase, si suponemos que las palabras de un documento son independientes podemos calcular:

$$P(D|class) = \prod_{w_i \epsilon D} P(wi|class)$$

- $P(wi|class)$ es la cantidad de veces que la palabra aparece en la clase debido a la suma total de todas las frecuencias de todas las palabras de la clase.
- Para evitar el underflow lo que podemos hacer es:

$$P(D|class) = \sum_{w_i \epsilon D} log(P(wi|class))$$

Problema de la probabilidad cero

Si la palabra de un documento no ocurre en ninguna de nuestras clases la probabilidad de P(palabra nueva — class) = 0 para cualquier clase. Por lo tanto, independientemente de todas las otras probablidadades vamos a decir que P(D—class) es cero para todas las clases y esto está **muy mal**.

Este problema se resuelve con la **correccion de Laplace**. Suponemos que la frecuencia de palabras nuevas es 1 y al cálculo de las probablidades siempre le agregamos la cantidad de palabras del vocabulario. Esto es:

$$P(w_i|class) = \frac{cant\ de\ apariciones\ de\ w_i}{suma\ de\ frecuencias\ de\ todas\ las\ palabras\ de\ la\ clase + cant\ de\ pal\ del\ voc}$$

Bayes puede ser víctima de **Overfitting**. Si existe una palabra rara que sólo aparece en una clase y en ningúna otra entonces cada vez que Bayes veea esta palabra va a clasificar automáticamente al documento dentro de la clase , el algoritmo alucina que cualquier cosa que tiene esa palabra pertenece a la única clase en la que la observó. Una posible solución es ignorar las palabras que tienen una frecuencia baja o que solo se observaron en una clase.

Perceptrón

Es un algoritmo de clasificación lineal. La teoría era que una neurona podía programarse como una salida binaria que dependia del resultado de la combinación lineal entre los inputs y los pesos asignados a los mismos. La neurona respondía 0 o 1 según el resultado de la combinación lineal fuese mayor o menor a un cierto umbral.

La **función de activación** debe emitir un resultado binario que puede ser 1 o 0, -1 o 1, etc.

Perceptrón puede utilizarse para la clasificación binaria. Es un algoritmo que **encuentra un hiperplano separador y no más que esto por lo que dista bastante de emular cualquier cerebro.**

Algoritmo Base

El algoritmo mediante el cual perceptrón .ªprende"los valores de w es:

- El preceptrón base usa 1 o -1 como clases.
- La predicción se hace calculando w * x.
- La función de activación es la función signo, es decir -1 si w*x es menor a 0 o 1 en otro caso.
- La regla de actualización es w = w + x*y que sólo aplicamos cuando nuestra predicción falla. Notemos que y nos da el signo para la actualización de w.

$$w = w + y_i x_i$$

Algoritmo mejorado

En el algoritmo mejorado agregamos un factor de aprendizaje para ir actualizando el valor de w de forma gradual y además permitimos que y tome dos valores cualesquiera y podemos usar cualquier función de activación.

$$w = w + \alpha y_i x_i$$

Es un algoritmo *online*, procesa los puntos uno por uno y nunca necesita memoria para más de un punto a la vez.

Los pesos soo se actualizan si el algoritmo está clasificando mal el punto, si la predicción es correcta el lgoritmo no hace nada. El algoritmo termina cuando la cantidad de errores es cero o bien cuando se han realizado la cantidad de iteraciones que deseamos hacer.

Normalización Online

En algunos casos es conveniente para que la convergencia sea más rápida normalizar todas las columnas de nuestros datos para que tengan promedio cero y desvio standar 1. Sin embargo esto no puede hacerse en un algoritmo online o que procese streams porque no coocemos la totalidad d elos datos. Una forma de simular la normalización de los datos es usar 1+log(x) en lugar de x, es decir tomar 1 + el logaritmo del dato para la columna que nos interese. De esta forma si una columna de nuestros datos tiene una escala completamente diferente a otra la diferencia se reduce notalbemente por el efecto del logaritmo.

Predicción

Una vez que perceptrón ha aprendido la recta separadora es muy simple clasificar cualquier punto nuevo, simplemente multiplicamos por w (a veces lo llamamos θ) y aplicamos al resultado la función de activación.

Característica importante: su capacidad de encontrar siempre una separación lineal, si es que existe y converger rápidamente a la solución.

Funciones de activación

- Función signo: la que vimos anterior mente.
- Función sigmoidea: $f(x) = \frac{1}{1+e^{-x}}$. Todos los valores menores a cero generan un númreo cercano a cero y los mayores a cero generan un número cercano a 1. El resultado de la función sigmodea puede considerarse una probabilidad. La probabilidad de que un cierto punto pertenezca a una cierta clase.
  La derivada es: $f'(x) = f(x)(1 - F(x))$ Esto hace que la función sigmoidea sea eficiente en algoritmos que usan el gradiente de funciones de multiples variables.

- Función tangente hiperbólica: es igual que antes pero más abrupta. Su derivada es $f'(x) = 1 - f(x)^2$

Perceptrón multiclase

Perceptrón es por definición un **clasificador binario** pero podemos adaptarlo o extenderlo a problemas de clasificación multiclase. Estas alternativas de extensión pueden aplicarse a otros algoritmos de clasificación binaria.

- **One vs All** vamos a entrenar perceptrón por cada clase (1: pertenece, 0: no pertenece) y luego cuando queremos clasificar und ato nuevo aplicamos todos los vectores aprendidos, el resultado mayor no indica a que clase pertenece el dato.

  $$c = argmax_y(w_y x)$$

  En este caso no aplicamos la función de activación sino que simplemente dejamos expresado el resultado w*x y luego teniendo todos los resultados tomamos el mayor para predecir la clase.

- **One vs One**: Vamos a aplicar perceptrón binario a cada par de clases posibles. Con todos estos clasificadores cada vez que queremos clasificar un dato aplicamos diferentes perceptrones y luego elegimos la clase que más triunfos tuvo. Es un poco más robusto que One vs All pero requiere de mayor cantidad de recursos, si las clases son muchas One vs One tiene que constuir una cantidad combinatoria de clasificadores y es inviable por lo que optaríamos por One vs All.

Conclusiones

- Es un clasificador lineal extremadamente simple.
- Funciona por definición de forma online pudiendo porcesar streams o datos masivos sin problemas.
- Es particularmente bueno para procesar datos binarios muy dispersos en un espacio dimensional enorme.
- Es una excelente opción para clasificar textos de forma muy eficiente.

SVM

Es un algoritmo de clasificación lineal, es decir que la salida del mismo será un vector n-dimensional. La idea de este algoritmo es muy simple y podemos expresarla a partir de lo que sabemos de perceptrón. En perceptrón buscabamos algún hiperplano que separe las clases que queremos clasificar. En SVM el objetivo es encontrar el mejor hiperplano.

En SVM el mejor hiperplano es aquel que maximiza el margen entre las clases. Es decir aquel que logra la máxima separación entre las clases. EDe todos los separadores posibles hay uno solo que cumple con esta condición.

Lo que vamos a agregar para ir transformando Perceptrón en SVM es el concepto de márgen. Cuanto más grande el margen más seguros vamos a estar sobre la clasificación del punto.

Algo muy interesante que tenemos que observar es que SVM se basa en encontrar el hiperplano para el cual el margen de separación mínimo es máximo. Esto implica que algunos de nuestros puntos van a quedar sobre los márgenes, estos puntos son lo que llamamos Suport Vectors y son

los que derinen el nombre del algoritmo. Es muy importante destacar que SVM sólo necestia de estos puntos para definir el hiperplano separador y todos los demás puntos los podríamos ignorar completamente.

La cantidad de support vectors debería ser cercana a la cantidad de dimensiones de los datos n+1.

Algoritmo SMO (sequential Minimal Optimization

Es un algoritmo muy simple para resolver SVM basado en Coordinate Ascent, la idea básica es ir optimizando un α_i por vez. ... WTF

Soft Margin

Cuando los datos no eran linealmente separables Perceptrón intentaba minimizar la cantidad de datos mal clasificados. En la simpleza de Perceptrón cualquier hiperplano con igual cantidad de puntos mal clasificados era igual en cuanto a la solución del problema. Ocurre que no sólo es bueno clasificar bien los puntos sino que aveces puede ser bueno clasificar mal algunos puntos. La idea es entonces introducir en SVM una penalización por cada punto mal clasificado: $y_i(wx_i + b) \geq 1 - \epsilon_i$ Donde x_i es el error que cometemos para el punto i es decir que está dentro del margen. ϵ_i será cero para todos los puntos por fuera del margen y un cierto valor positivo para los puntos dentro del margen.

Agregamos ahora un costo constante C para penalizar estos errores. El planteo es:

$$min_{w,b}\frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\epsilon_i$$

$$y_i(wx_i + b) \geq 1 - \epsilon_i$$

C regula si el algrotimo va a intentar minimizar la cantidad de puntos dentro del margen o bien maximizar el margen entre las clases. Con un margen muy amplio tendremos muchos puntos dentro del mismo y con un margen muy pqueño muy pocos, en general queremos encontrar el equilibrio. Notemos que agregando el concepto de soft-margin el único cambio que tenemos en nuestro planteo es que los que antes estaban restringidos a ser positivo ahora tienen que ser también menores o iguales al factor de penalización C.

The C parameter trades off correct classification of training examples against maximization of the decision function's margin. For larger values of C, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. A lower C will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy. In other words"C" behaves as a regularization parameter in the SVM.

The kernel trick

Es un mecanismo muy ingenioso que nos permite clasificar datos que no son linealmente separbles con un clasificador lineal. El truco está en el Teorema de cover:

Teorema: Dado un set de datos que no es linealmente separable es posible con una alta probabilidad de transformarlo en un set de datos linealmente separable proyectándolo en un espacio de más dimensiones mediante alguna tranformación no lineal.

La función de mapeo la vamos a llamar $\phi(x)$ que recibe un punto x en n dimensiones y devuelve otro en x en una cantidad de dimensiones mayor. Ahora debemos notar que en SVM en ningún momento necesitamos acceder a los puntos de nuestro conjunto de datos sino simplemete al

producto interno entre los mismo. Por eso en SMO SVM cuando los datos son pocos pre-calculamos la matriz K de puntos internos. Si agregamos la funcón de mapeo al producto interno tenemos lo que se llama un kernel.

$$K(x, y) =< \phi(x), \phi(y) >$$

El "kernel trickçonsiste en calcular el resultado de K(x,y) sin realizar el mapeo de los datos, es decir directamente a partir de x e y calcular cuál será el resultado de su producto interno luego de una cierta transformación en que lleva a los puntos a más dimensiones, repetimos, sin realizar la transformacíon en ningún momento.

Kernel Polinómico

$$K(x, y) = (< x, y > +c)^d$$

Es decir que hacer el calculo anterior obtnemos el producto interno de los puntos en una dimensión mayor sin hacer el mapeo a la dimensión mayor.
Para que sea un kernel tiene que ser posible hallar la función tal que el resultado del kernel sea igual al producto interno entre los dos vectores mapeados por ϕ(x).

Para llegar a una definición necesitamos definir una matriz de kernel. Sea K(x,y) un kernel y sea X un conjunto de m puntos en n dimensiones, definimos a la matrix K como la matriz en la cual $K_{i,j} = K(x_i, x_j)$. Esta matriz es muy parecida a la matriz de productos nternos que podemos precalcular en SVM solo que SVM lineal $K(x, y) =< x, y >$ Es decir que el kernel lineal es el producto interno.

Teorema de Mercer Sea $K : R^n -> R^n$ una función. Decimos que K es un kernel válido si y sólo si para cualquier conjunto de puntos $x_1, x_2, ..., x_m$ pertenecientes a R^n la matriz asocciada a K es simétrica y positiva semi definida.

Destaquemos que el hecho de que la matriz sea simétrica y positiva semidefinida es condición necesaria y suficiente, es decir que si verificamos estas condiciones en algún kernel entonces sambemos que existe el mapeo $\phi(x)$ sin necesidad de calcularlo.

Kernel Gaussiano RBF

$$K(x, y) = e^{-\frac{||x-y||^2}{2\sigma^2}}$$

Sigma es un hiperaprámetro que depende de los datos a los cuales aplicamos el kernes. Esta es una función de semejanza muy usada que da como resultado algo muy cercano a 1 cuando x e y están muy lejos y da como resultado algo cercano a 1 cuando x e y son muy similares.

Este Kernel cumple con las condiciones de Mercer por lo que sabemos que es un kernel válido y que por lo tanto existe una función de mappeo, en este caso la función mapea a los vextres x e y a un espacio de dimensiones infinitas., pno podemos calcular $\phi(x)$ pero sabemos que existe por el Teorema de Mercer.

Aproximación de Nystrom

Uno de los problemas al usar kernels es la necesidad de calcular la matriz del kernel que si los puntos son muchos puede ser potencialmente enorme. Cuando el cálculo de la matriz completa no es factible aparece la aproximación de Nystrom como una solución. Los pasos son los siguiente:

- Elegir al azar una cierta cantidad de puntos de nuestro conjunto de datos.

- Calcular la matriz del Kernel para estos puntos (siendo la cantidad un número manejable).

Cuando queremos calcular K(x,y) si los puntos están en la matriz entonces el resultado es automático. SI los puntos no están en la matriz entonces hay que calcular los puntos de la matriz mas cercanos a x, los puntos más cercanos a y y luego estimar K(x,y) como el promedio ponderado de aplicar K a los más cercanos.

Kernel SVM

El objetivo de esta sección es hablar sobre los hiperparámetros de Svm usando un kernel RBF, tenemos dos hiperparátros a optimizar que son: C costo de clasificar mal usando soft margin y σ que es el radio de la función de RBF para el kernel gaussiano.

SVM con kernel RBF es capaz de encontrar una frontera no lineal para separar las clases. En líneas generales este algoritmo puede encontrar prácticamente cualqueir tipo de frontera, igual que KNN pero con la ventaja de no solo usar los puntos cercanos a la frontera para clasificar es decir los support vectores.

Conclusiones

SVM es un pariente cercano de KNN porque se basa en la semejanza (kernel) entre los puntos para determinar la frontera entre las clases, pero el comportamiento de ambos algoritmos es totalmente diferente. KNN tiene ocmo problema la maldición de la dimensionalidad meintras que SVM brilla cuando podemos proyectar los datos en infinitas dimensiones para encontrar una separación lineal. Es uno de los pocos algoritmos que navega por hiper-espacios de enorme dimensionalidad de manera confiable.

Clustering

El proceso que se conoce como clustering es el ejemplo más claro de **aprendizajo no supervisado**. Dado un set de datos se quiere agupar los datos en clusters de forma tal que todos los datos dentro de un mismo cluster sean similares entre sí pero diferentes a los de cualquier otro cluster. Tiene la característica de poder trabajar sin la necesidad de labels.

Clustering Jerárquico

El método de clustering Jerárquico es uno de los algoritmos de clustering más importantes ya que genera una descomposición jerárquica del set de datos creando clusters desd k=1 hasta m, siendo m la cantidad de puntos del set de datos. Es util tanto como heramienta de clustering así tambien como instrumento para el análisis exploratorio de los datos.

El gran problema del clustering jerárquico es su eficiencia para grandes volúmenes de datos.

El algoritmo es así:

- Comienza suponiendo que cada punto es un cluster. En el primer paso hay m clusters.
- Luego en cada paso se toman los ds clusteres más sercanos entre si y se los une en un nuevo cluster.
- Este proceso se repite hasta que queda un único cluster.

Hay dos cuestiones importantes que es necesario definir para poner en archa el algoritmo:

1. Distancia a utilizar. Depende de la naturaleza de los datos.

2. Forma de calcular la distancia entre clusters:

   a) La distancia mínima entre dos puntos de cada cluster.

   b) La distancia máxima entre dos puntos de cada cluster.

   c) El promedio de las distancias de todos los puntos de un cluster contra todos los puntos del otro cluster.

   d) La distancia entre los promedios de los puntos de cada cluster (centroides).

   e) El método de Ward.

Encontrar la cantidad de clusters

Entender qué cantidad de clusters elegir no es una tarea trivial. Al aplicar el algoritmo completo (es decir, hasta que queda un único cluster con todos los puntos) se obtiene como resultado una lista que indica qué clusters ebemos unir en cada paso. En cada paso se puede indicar la distancia entre los clusters.

Este método es sensible a los outliers

A partir de los datos generados por la función de clustering jerárquico, se puede generar un diagrama que nos muestre en cada paso qué clusteres se han unido y con qué distancia. Este diagrama se lo conoce como **Dendrograma**.

Performance

El método de clustering jerárquico es extremadamente poderoso puesto que se tienen todas las conclusiones de clustering posibles para diferentes valores de k al alcance de la mano. Además tiene información valiosa en la descomposición jerárquica del set de datos.

Como en cada caso encontrar los clusters que están a distancia mínima implica recorrer los n clusters (fuerza bruta) puede implementarse usando LSH. En tal caso la solución pasa por aproximar el problema de los clusters más cercanos en cada etapa, en genera, se puede usar LSH como un algoritmo para evitar tener que calcular la distancia de todos contra todos. De esta forma, por cada clustersimplemente se busca el más cercano entre la lista de candidatos devuelto por LSH. Esto puede usarse calculando la distancia entre clusters como la distancia entre controides. Cada vez que se crea un nuevo cluster se promedian sus puntos y este centroide será el punto que se hasheará con LSH insertándolo en la estructura de datos correspondiente. Luego por cada punto se calcula el más cercano y se obtiene el par que de distancia mínima.

La gran ventaja se usar LSH es que buscar en un cluster más cercano a uno dado es O(1) en lugar de O(n).

Una solución por fuerza bruta de clustering jerárquico puede ser mejor que K-Means, una solución aproximada usando LSH en general puede ser mejor igual o pero que K-means y por eso es que K-Means suele ser el método de clustering más usado.

K-Means

K-Means es uno de los más importantes de los algoritmos de clustering, no sólo com algoritmo de lcustering sino como parte de otros algoritmos.

En K-Means la cantida de cluster k es siempre un hiperparámetro. Lo principal es entender lo que significa *la mejor forma posible*. Se puede definir a cada cluster a partir de su centro (centroide) en cuyo caso la mejor distribución posible es aquella que minimiza la distancia entre cada punto

y el centroide que se le ha asignado. Los centroides tienen igual cantidad de dimensiones que los puntos y pueden o no coicidir con los puntos de los datos. Un centroide que tambié es un punto de los datos se llama clusteroid. En generar no se va a pedir que los centroides sean también puntos.

Entonces, es importante entender esta función para compredndner el problema genérico de clustering. Se sabe que hay k centroides y que cada uno de estos centroides puede estar en cualquier punto del espacio. El objetivo es encontrar la posición optima para estos centroides de forma tal de minimizar la distancia total entre los puntos y los centroides que se le han asignado. Es evidente que cada punto debe estar asignado a su centroide más cercano para minimizar la distancia por lo que el problema puede resumirse a ecnontrar la posición optima para los k centroides.

Se puede observar que los mínimos locales son muy cercanos al mínimo global por lo que un algoritmo que sea localmente óptimo no está lejos de ser globalmente óptimo. Esta propiedad es la que aprovechó Llyld para desarrollar K-means.

El algoritmo de Loyld que es el que se conoce como K-Means es una forma de aproximar el problema de clustering generar. Una de sus grandes virtudes es que se trata de un algoritmo muy sensillo que consta de solamnete un ciclo y dos operaciones. Lo que hace es inicializar k centroides al azar, luego se asigna cada punto a su centroide más cercano. Luego se recalculan los centroides como el promedio de todos sus puntos y se repite el proceso hasta que los centroides converge.

K-Means pertenece a la familia de algoritmo EM (expectation maximization), partiendo de una cierta posición para los centroides la fase E (expectation) asigna a cada punto el centroide más cercano. La fase M (maximization) recalcula cada centroide de forma tal de minimizar la distancia desde el centroide a los puntos asignados al mismo.

La convergencia puede verificarse mediante la diferencia entre los centroides entre el paso anterior y el actual. Cuando los centroides prácticamente ya no se mueven se declara la convergencia del algoritmo.

K-Means ++

Es una variante de K-means en donde lo único que cambia con respecto al algoritmo Lloyd es la forma en que se inicializan los centroides. La idea de K-Means ++ es asignar los centroides de forma espaciada, de esta forma el óptimo local obtenido por K-means tiene mayor probabilidad de estar cerca del máximo global; o lo que es lo mismo el óptimo local será lejos de los óptimos que son muy malos con una probablidad alta. El algoritmo se puede definir de la siguiente manera:

- Elegir un punto al azar como primer centroide.
- Por cada punto calcular la distancia con cada centroide y quedarnos con el mínimo.
- Calcular la probablidaidad de cada punto como su distancia mínima dividido la suma de todas las distancias. Notar que si ya se tiene más de un centroide entonces la probabilidad se calcularía en base a la distancia de cada punto al centroide más cercano.
- Elegir un nuevo punto al azar en base a estas probablidades.
- agregar el punto a la lista de centroides.

El problema de K-Means ++ es que para datos realmente masivos necesita hacer k iteraciones sobre los datos para poder elegir los centroides, tarea que todavía es más inificiente cuando k es un número largo.

Existe una variante a K-Means ++ que es K-Means Paralel. La idea es hacer sólo log(k) pasadas sobre los datos, en cada iteración se agregan L centroides en lugar de 1 y el resultado es que

se tendrá unca cantidad de centroides mayor a k. Los centroides finales se obtienen clusterizando los candisatos a centroides ponderados por la probablidad con la que fueron seleccionados, este paso se puede hacer, por ejemplo, usando clustering jerárquico.

K-Means ++ y su variante K-Means Paralell aceleran la convergencia del algoritmo.

El valor de K a utilizar

En general, la forma de buscar k es mediante Grid-Search con diferentes valores. Como métrica para evualar la calidad de cada valor de k se puede usar un promedio de nuestra función de distorición para varias iteraciones de K-Means o el díametro promedio de los clusters, tomando como diámetro de un cluster la distancia máxima entre dos puntos del mismo.

Es de destacar que cuando K es igual a la cantidad de puntos el costo total es cero y cada cluster contiene sólo un punto pero esto obviamente no tiene mucho sentido.

Función de distorción: $J(c, idx) = \sum_{i=1}^{n} ||x_i - c_{idx}||^2$

Diagramas de Vonoroi

El proceso de clustering o de K-Means genera lo que se conoce como diagrama de Vonoroi o un ateselación de Vonoroi. Este diagrama consiste en delimitar la frontera hasta la cual llega cada centroide.

El diagrama de Vonoroi muestra a queé cuslter quedaría asignado un punto nuevo que se quisiera agregar al set de datos sin ocrrer de nuevo K-Means. Un detalle fundamental de esto es que, como se puede ver, K-Mean realiza una partidicón de todo el espacio n-dimensional. Esto quiere decir que se puede representar a cualquier punto del espacio de n dimensiones en k dimensiones de acuerdo a la separación establecida por el diagrama de Voronoi correspondiente al resultado de K-Means.

Notar la forma que toma un diagrama de Vonoroi, cada sector del diagrama depende única y exclusivamente de un punto: el centroide. Las fronteras indican la separación entre centroides, a cada lado de as líneas estamos más cerca de uno u otro centroide. EL tamaño de cada área no depende de la cantidad de puntos asociados al centroide sino de que tan aislado se encuentra el centroide respecto del resto, cuanto más ailado más grande será la superficie de su área.

K-Means Online

La versión online porcesa un punto por vez del set de datos, no necesita conocer todos los puntos antes de empzar y requiere un mínimo uso de memoria.

La idea del algoritmo es:

A partir de k centroides inicializados de alguna forma que es externa al algoritmo se procesa cada punto asignándolo al centroide más cercano y luego se mueve ese centroide hacia el punto procesado de forma proporcional a la cantidad de puntos que tiene el cluster en total. Es necesario contar con un vector de k elementos en donde se irán contando cuántos contiene cada cluster hasta el momento.

Es importante destacar que por cada punto del algoritmo se computa la distancia contra cada centroide pero la cantidad de centroides nunca es demasiado grande. El algoritmo tradicional, en cambio, computa en cada iteración la distancia de todos los puntos a todos los centroides.

La versión online de K-Means únicamente asigna una vez por punto, una vez que el punto fue asignado a un centroide permanece dentro de dicho cluster. Esto puede solucionarse realizando más de una iteración. En la segunda iteración los puntos que quedaron mal asignados en la primera pasada pasan al cluster que les corresponde.

Lo más destacable es que a partir de tres o cuatro iteraciones el resultado de la versión online es siempre mejor que la versión traidicional.

La versión de K-Means++ es mejor que las dos anteriores aunque muy similar a la versión online con múltiples iteraciones con el costo de tener que pre-procesar los datos para lograr los centroides iniciales.

Soft K-Means

Es una generalización de K-Means que se conoce como Fuzzy C-Means o EM (expectation maximization). El objetivo de este algoritmo es: permitirle a cada punto pertenecer al mismo tiempo a más de un cluster. Cada punto pertenece a todos los cluters y lo que determinamos en el algoritmo es el grado de pertencencia de cada punto a cada cluster o la probabilidad de que el punto pertenezca a cada cluster.

Se dice que la asignación de K-Means es hard ya que cada punto queda asignado a un único cluster (al más cercano) y la posición de todos los demás centroides no es importante.

En la versión soft se asigna a cada punto k probabilidades, la probablidad de que el punto se asigne a cada uno de los k-centroides. Para calcular estas probablidades se pueden sumar todas las distancias y luego realizar $1 - \frac{distancias}{suma}$.

Pasos del algoritmo:

- Dada la asiganción de cada punto a cada cluster calcular los centroides.
- Dados los centroides calcular la asignación de cada punto a cada cluster.

Estos procesos se repiten en un ciclo hasta que los centroides no cambien (con una cierta tolerancia).

Para calcular la asignación de cada punto i a cada cluster k se usa la siguiente fórmula:

$$w_{ik} = (\sum_j \frac{d(c_k, x_i)^2}{d(c_j, x_i)^2})^{-1}$$

Para calcuar los centroide se usa la siguiente fórmula:

$$c_k = \frac{\sum_i x w_{xk}^2}{\sum_x w_{xk}^2}$$

Es decir, un promedio ponedrado de todos los puntos por el peso de la asignación de cada centroide.

Versión Online

Se toma un punto, se calcula la probabilidad de que cada centroide en base al punto y se acerca a cada centroide en proporción a dicha probabilidad y a la acumulación total de probabilidades que tenga el centroide. Es decir que cuantos más puntos se peguen fuertemente a un centroide menos se moverá el mismo al procesar un punto nuevo.

Esta version tiende a generar casi los mismo resultados que la anterior pero con menos iteraciones.

La base de Kernerls y sus aplicaciónes

Supongamos que tenemos X donde sus filas son puntos y sus columnas cada dimensión. Y tenemos IDX en donde sus filas son el grado de pertenencia de cada punto y las columnas la cantidad de clusters (k). Luego C es la matriz donde sus filas con las coordenadas de los centroides.

Lo interesante de todo esto es que se pueden tomar a los centroides como una base y a las asignación como coordenadas y entonces la matriz de puntos X se puede aproximar como IDX * C.

En el caso de K-Means tradicional el funcionamiento es igual pero IDX tiene únicamente 1s y 0s porque la asignación es hard pero igual podemos usar esta matriz y la matriz de clusters para aproximar la matriz original. En el caso de soft K-Means cada punto se representa mediante una combinación lineal de todos los centroides.

Conclusiones a las que se llega:

- Se puede usar K-Means como una forma de reducción de dimensiones, en donde cada punto se representa mediente las coordenadas del mismo respecto a una base que son los centroides.
- La cantidad de cluster (k) da la cantidad de vectores en la base y por lo tanto permite regular la cantidad de dimensiones en la que representamos a los datos.
- El diagrama de Voronoi da las fronteras para representar nuevos puntos de acuerdo a la base obtenida originalment por K-Means.
- Cada vector de la base es un centroide por lo que tanto es un promedio de puntos de nuestro set de datos.
- Por lo tanto usando K-Means como base se representa a cada punto mediante una combinación lineal de vectores que surgen de promediar los puntos del set de datos.
- Por ser promedio de puntos del set de datos los vectores de la base de K-Means pueden interpretaarse como nuevos datos.
- A mayor cantidad de centroides más precisa será la reconstrucción de la matriz de datos original.
- Cada punto se representa mediante una combinación a fín de centroides (las coordenadas son positivas y suman 1) es decir que por cada centroide se toma un porcentaje del mismo y se lo usa para representar parte del punto. Notar que esto vale tanto para soft K-Means como para hard K-Means en cuyo caso hay un único centroide con coordenada 1 y todos los demás 0.

Reducción de dimensiones con K-means

Para reducir dimensiones una posiblidad es representar a cada punto mediante su centroide más cercano (hard K-Means). Los sistemas basadon en coordenadas requieren de la base de K-Means para tener sentido, el objetivo es encontrar una forma de representar los datos en k dimensiones sin que sean necesarios los centroides.

The K-Means Trick Una forma de hacer esto es por cada punto calcular la distancia a cada uno de los k centroides y usar estas k distancias como k coordenadas por cada punto. Esto da una una forma de respresentar simple y directamente los datos en k dimensiones.

Para corregir porblemas de escala se puede usar una función que mapee la distancia al centroide x a un número entre 0 y 1 por ejemplo: $y = e^{-x\gamma}$. En donde γ es un hiperparámetro. Usando grid-search y luego un histograma de los valores que tenemos para las k coordenadas de nuestros

puntos buscando que la distribución tienda a ser notrmal. Tener features con distribución normal es algo beneficioso para varios algoritmos y se lo puede lograr usando K-Means.

Es muy sorpendete que estos features aprendidos por K-Means funcionan realmente muy bien como representación del set de datos. Es muy común que un algoritmo de clasificación funcione mejor con estos features que con los datos originales.

Dado que K-Meanss es en general eficiente un método sorpendete para manejar clasificación en un set de datos es aplicar K-Means para reducir los datos a k dimensiones y luego aplicar sobre estos datos un clasificador lineal. hay cosas que funcionan muy bien basadas en esta técnica.

Kmeans Esférico

K-Means esférico es la aplicación de K-Means a la distancia angular en lugar de la distancia euclidiana. Esta distancia es útil siempre y cuando se agrupen los datos en base a la dirección hacia la cual los datos apuntan en lugar de su posición en el espacio.

Para aplicar K-Means esférico, el primer paso es normalizar todos los vectores para que tenga norma 1. Esta normalización se realiza por filas. Los centroides iniciales se generan al azar con componentes entre 0 y 1 y también se normaliza por filas. Ahora que se tienen un conjunto de m puntos en n dimensiones y hay k centroides en n dimensiones, para calcular cuál es el centroide más cercano a cada punto hoy que hacer el producto interno entre cada punto y cada centroide. Se pueden hacer todas las cuentas calcluando el producto entre la matriz X de mxn y la matriz de centroides kxn transpuesta. Este producto nos devuelve una matriz D de m filas por k columnas, donde cada fila representa a un punto y cada columna representa la semejanza (coseno) a cada uno de los k centroides. Para determinar a qué sentroide se asigna cada punto simplemente se observa cuál es el valor máximo para los cosenos entre cada punto y cada centroide. Notar que al contrario de Kmeans tradicional aquí se toma el máximo producto interno que corresponde a la mínima distancia angular.

La matriz de asignación de puntos a centroides S se la contruye ce m+k en donde cada fila sólo tiene un valor distinto de cero que es el valor del coseno entre dicho punto y su centroide más similar. Es decir, S es la matriz en la cual por cada fila de D se deja únicamente el valor máximo y el resto se los convierte en 0.

Se pueden recuperar los centroides de la siguiente manera: $C = S^t X$ o $C = C + S^t X$

Esto se conoce como Dampening y sirve para que los centroides no cambien tan bruscamente de una iteración a otra.

El paso final de K-Means esférico es normalizar los centroides para que tengan norma 1 y se pueda volver a usar en la próxima iteración.

Este algoritmo es más veloz que la variante Euclideana.

Kernel K-Means

Es la aplicación de un kernel al algoritmo de K-Means. De esta forma se puede lograr clustering cuando los clusters no son linealmente separables. El funcionamiento y objetivo del kernel es el mismo que se utiliza con SVM: transformar los datos a un espacio de mayor cantidad de dimensiones, de forma que sean linealmente separables. En este caso se quiere que los cluster queden separados linealmente puesto que es un requisito para K-Means.

K-Modes

Es una variante de K-Means para datos que tienen atributos categóricos. No se necesita convertir los datos (ej: one hot encoding).

El algoritmo en sí es identico al de antes sólo hay que redefinir de qué forma se computa el promedio de punto para calcluar los centroides y de qué forma se miden la distancias entre puntos y centroides. para medir la distancia entre dos puntos se usará la norma l_0, es decir, la cantidad de atributos en los cuales los datos difieren.

Para calcular el prmedio entre n puntos se tomará por cada dimensión el valor que más veces aparece, es decir, la moda. Si hay empate se desempata al azar.

Esto funciona bien cuando hay muchos atributos y por lo tanto mayor precisión en el rango de distancias posibles.

K-Means y el problema de los vecinos más cercanos

Es posible usar K-Means para aproximar el problema de los vecinos más cercanos (KNN) de forma eficiente.

Para buscar los puntos más cercanos a un cierto punto q hay que comparar q contra cada uno de los k centroides y luego para el centroide más cercano comprar contra todos los centroides asignados a dicho punto.

El proceso se puede hacer online usando K-Means Online e ir conslutando a medida que se agregan los datos, lo cual sirve como un algoritmo de Streaming.

Si el punto pertenece a la frontera del cluster entocnes es posible que estemos dejando de lado algunos puntos que pueden ser similares. Por lo tanto, se podría buscar en más de un cluster, es una negociación entre precisión y performance.

Datos Masivos, realmente masivos

Si los puntos son tantos que la cantidad de puntos a buscar por cada centroide se hace muy grande una posible solución es usar un valor de k más grande para tener menor cantidad de puntos asignados a cada centroide pero es evidente que esta solución no es eficiente porque habrá que comprar el query contra una cantidad mayor de centroides y además el proceso de K-Means se vuelve más lento.

Hay dos soluciones a este problema: una basada en una estructura llamada inverted multi index y otra usando árboles basados en K-means.

The inverted Multi-Index

K-means Trees

Clustering Espectral

Surge originalmente como una forma de realizar cortes entre componentes de un grafo. Al igual que Kernel K-Means tiene la capacidad de detectar formas de clusters de formas arbitraris, no lineales.

Cuando los clusters tiene forma no-convexas o cuando las distancias son binarias (redes sociales) K-Means falla.

Matriz de afinidad: $W_{i,j} = e^{\frac{-||x_i - x_j||^2}{2\sigma^2}}$. Lo qe indica es el parecido entre x_i e x_j que son dos puntos (en el numerador del exponente se puede usar cualquier distancia).

La matriz D es una matriz diagonal que tiene la suma de las filas de la matriz W.

Luego, se define L como (elegir una opción):

- $L = D - W$
- $L = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$
- $L = D^{-}1 - W$

Ciertas propiedades de la teoría de grafos dicen que el espectro de la matriz Laplaciana sirve para encontrar cortes mínimos en un grafo. Un corte mínimo es aquel que divide el grafo en dos conjuntos de nodos de la forma tal que las aristas entre nodos de los dos conjuntos tienen afinidad mínima. Es decir que se lo puede utilizar para separar el grafo en componentes lo cual equivale a realizar clustering entre los puntos que se han representado como grafos del nodo.

Propiedad interesante: la cantidad de autovalores de la matriz Laplaciana que son iguales a cero es igual a la cantidad de componentes conexas en el grafo (L representa la conectividad de los nodos).

Luego con los m autovectores *menos significativos* construir una matriz: U, normalizar las filas de U, aplicarke K-Means y por último, para cada punto de X asociarle el cluster que le asignamos a la fila correspondiente en U-Normalizada.

Aproximación KASP

Clustering Espectral tiene como problema la necesidad de encontrar con la matriz de afinidad W que es una matriz de mxm sien m la cantidad de datos. Esto puede llegar a ser un número imposible de manejar. Entonce se realiza la siguiente aproximación que permite usar clustering sin calcular W de manera completa:

- Usar K-Means para los datos originales para generar z centroides.
- Usar clustering espectral sobre los centroides.
- Asignar a cada punto del set original el cluster al que pertenece su centroide.

Es notable que se usa K-means como método de compresión del set de datos m a k puntos en donde k puede ser arbitrario.

Clustering Espectral puede aplicarse de forma directa a la detección de comunidades en grafos.

CURE

Clustering Using Representatives es un algoritmo de lcustering para grandes cantidades de datos, que es robusto a datos anómalos y permite clusters de forma arbitraria. CURE combina clustering kerárquico y kmeans de forma eficiente.

- El primer paso es elegir un subconjunto de puntos al azar y agruparlos en k cluster usando clustering jerárquico.

- El segundo paso es de cada cluster formado elegir f puntos representativos que son f los puntos del cluster que están más alejados entre sí.
- El tercer paso es mover los puntos represetntativos un 20 porciento hacia el centro del cluster.
- El Ultimo paso es por cada punto del set de datos que falta procesar se lo asigna al cluster que corresponde al punto representativo más cercano al punto.

El orden de Cure es de $O(n^2 logn)$.

DBScan

Algoritmo de clustering basado en el concepto de densiodad, es decir, encontrar zonas en las cluaes haya una densidad e puntos altas e identificar estas zonas como un cluster. Es capaz de detrminar automáticamente la cantidad de clusters y puede detectar outliers.

Hiperparámetros:

- e: La distancia mínima entre dos puntos para estar en un mismo cluster. En otras palabras la densidad con la cual se agruparán los puntos.
- k: La cantidad mínima de puntos vecinos a la distancia correcta para generar nuevos clusters k. En otras palabras, la cantidad mínima de puntos que puede tener un cluster.

Los puntos que no pertencen a un cluster y no son suficientes para formar un lcuster son los outlier.

El algoritmo comienza con un punto cluaquiera, si el punto no pertenece a un lcuster entonces se analiza si tiene al menos k -1 vecinos a distancia menor o igual a e en cuyo caso se forma un nuevo cluster con estos k puntos. Si el punto ya pertenece a un cluster entonces se buscan los puntos a distancia e o menor y se los agrega al cluster.

Es capaz de detectar formas arbitrarias de casi cualquier tipo. **Esto funciona siempre y cuando todos los cluster tengan densidades parecidas, es decir, que los puntos dentro de cada cluster se encuentren a distancias similares.** Cuando los lucster tienen distintas densidades DBsan falla ya que puede no ser posible enontrar un parámetro e que sirva.

En cuanto a **performance** se encuentra el problema de por cada punto encontrar los puntos a distancia menor o igual a e y eso es O(n^2) por fuerza bruta. La solución es usar algún tipo de índice espacial para llevar esto a O(n log n). Estos índices no escalan bien en muchas dimensiones por lo que en espacios de muchas dimensiones DBScan sufre la maldición de la dimensionalidad.

Problema fundamental de DBScan:

- Es bastante sensible a los parámetros.
- Es particularmente difícil estimar e.
- Como la densidad se define a priori, no maneja bien clusters con densidades muy distintas.
- Como es plano, no puede encontrar clusters más densos dentro de otros más grandes.

HDBScan

Algoritmo basado en DBScan que soluciona el problema de las densidades diferentes en los clusteres. El problema es que si se usa una distancia mi pequeña e los lcusteres con densidades de puntos bajas quedan sin identificar es decir como unenorme conjunto de ouliers mientras que si se usa una desidad grande suficiente para encontrar los lusteres menos densos empiezan a mergearse

entre si.

El primer paso de HDBScan es construir una matriz de distancias entre los puntos pero en lugar de usar la distancia euclideana en HDBScann se usa una distancia especial denominada Mutual Reachability Distance (MDR).

La **Distancia MDR** asocia a cada punto con la distancia de su vecindario que es la distancia máxima entre sus k vecinos más cercanos. k es un hiper-parámetro para el algoritmo. A esta distancia se la conoce como el $core_k$ del punto.

La distancia MDR se la define como: $MDR(a, b) = maxcore_k(a), corek(b), d(a, b)$.

MDR funciona particularmente bien junto con clustering jerárquico que es uno de los algoritmos que va a usar HDBScan.

El árbol generador mínimo

Usando la matriz de distancias MRD el objetivo de HDBScan es encontrar islas de puntos que tienen alta densidad en donde la densidad es variable para cada isla. Conceptualmente lo que se hace es considerar a los datos como un grafo en donde el peso de las aristas es la distancia MRD entre los puntos.

Una forma análoga de hacer esto es construyendo el árbol generador mínimo para el grafo. Esto se puede hacer con el algoritmo de Kruscal (se empieza con el grafo vacío y en cada paso se agrega la arista de menor peso que conecte el arbol actual a un nodo que todavía no forme parte del árbol). El árbol generado con MDR es distinto al árbol generado con distancia euclidea.

Una vez que se tiene el arbol generador mínimo se usa el **algoritmo de clustering jerárquico** sobre los puntos usando la distancia MRD.

Al igual que en DBScan se quiere fijar un tamaño mínimo para que un conjunto de puntos se pueda consierar un cluster. Para que un split del dendograma genere dos nuevos clusters éstos tienen que tener al menos la cantidad mínima de puntos que se estableció. En caso contrario se lo considera como un cluster que pierde puntos pero sigue siendo el mismo cluster.

Esto permite podar el dendograma en donde sólo quedan los splits que generan clusters válidos. La cantidad de puntos en los clusters puede ir variando porque los clsuters que no se subdividen pueden perder puntos.

Extraer los clusters

Intuitivamente se quieren aquellos clusters que persisten a los largo del tiempo puesto que los clusters que duran muy poco probablemente sean simplemente ruido. Observando el dendograma luego de la poda se puede decir que se quieren los clusters que tienen mayor área en el plot. Si se selecciona un cluster no se podrá luego elegir a ningún descendiente del mismo porque compartirían puntos.

En primer lugar se necesita una medida de distancia para medir la persistencia de los clusters. Recordar que en cada paso del clustering jerárquieco se tiene la distancia a la que estaban los clusters unidos. Para ello, se utilizará $\lambda = \frac{1}{distance}$. Por cada cluster se tendrán entonces dos lambdas: uno que el es valor con el cual el cluster nace y otro con el cual el cluster muere (es decir se subdivide en dos clusters de al menos p puntos). A estos se los llamarán λ_{birth} y λ_{death}.

Además, por cada punto p es un cluster se tiene λ_p que es el valor de λ en el momento en el cual el punto cayó fuera del cluster, y este valor va a estar entre λ_{birth} y λ_{death} ya que el punto o bien cae fuera durante la vida del cluster o bien abandona el cluster cuando el cluster muere. Notar que el hecho de que un punto caiga fuera de un cluster no quiere decir que el punto deje de pertenecer al cluster.

Se define como **estabilidad** de un cluster a: $\sum_{p \epsilon cluster}(\lambda_p - \lambda_{birth})$.

El algoritmo para seleccionar los clusters es: se empieza con cada hoja del dendograma podado como un clsuter seleccionado. Se recorre el árbol hacia arriba (ordenamiento topológico inverso). Si la suma de la estabilidad de los dos hijos es mayor que la estabilidad del padre entonces seteamos la estabilidad del padre como la suma de la estabilidad de los hijos. Si la suma de establilidad de los hijos es menor que la estabilidad del padre entonces se declara al cluster padre como un clsuter seleccionado y se des-seleccionan a los hijos. Una vez que se llega a la raíz el conjunto de cluster seleccionados constitye el grupo de clusters.

Una vez que se obtienen los cluster es fácil llegar al resultado final. Cualquier punto que no esté en alguno de los cluster seleccionado es ruido y se le asigna un label -1. Por cada cluster se tiene el valor de λ_p para cada punto del cluster y si se normalizan estos valores en el rango 0..1 se tendrá una media de pertenencia de cada punto al cluster. Esto se lo puede usar como la probabilidad de cada punto pertenezca al mismo.

Un punto a considerar es que HDBScan tiene a detectar más outlieres de los necesarios y esto puede corregirse controlando el porcentaje de outliers y variando los hiper-parámetros del algoritmo.

HDBScan es un algoritmo muy interesante ya que permite encontrar clusters de forma arbitraria y con densidades arbitrarias, identificar ouliers y detectar automáticamente el número de clusters.

Streaming

Un **stream** es un flujo de datos que no tiene fin. Los algoritmos tradicionales tienen graves problemas al procesar streams. En primer lugar es muy difícil calcluar estadísitcas, ranking y otras métricas interesantes si los datos cambian constantemente. Esto se debe a que **cuando calcularmos un cierto valor este ya ha perdido actualidad por la cantidad nueva de información recibida.** Por otro lado, ningún algoritmo puede procesar una cantidad de datos infinitos.

Los **algoritmos de streaming** trabajan en memoria y procesan los datos a medida que estos ocurren actualizando constantemente su resultado. Por lo tanto, podemos obtener datos en tiempo real de un determinado stream.

Reservoir Sampling

Es un algoritmo que permite convertir un stream en un conjunto de datos finitos. La idea es mantener una muetra (sample) del stream en memoria.. Luego podemos calcular los datos y estadísticas que nos interesen sobre el stream a partir de la muestra usando algoritmos tradicionales. El algoritmo necesita mantener en memoria una cantidad fija de elementos k. Para que las estadísiticas tenga sentido debemos garantizar que la probabilidad de que un dato del stream esté en la muestra sea igual a:

$$P(X_i \epsilon sample) = \frac{k}{N}$$

.

Donde N es la cantidad de elementos del stream hasta el momento. Notemos que la probablidad es dinámica y que la probablidad de todos los elementos independientemente de cuando los observamos deber ser igual.

El algoritmo funciona de la siguiente manera:

- Por cada dato que ingresa calculamos la robabilidad de ingresar.
- Generamos un número random y luego en función de este número random ($\epsilon[0,1]$) y de la probablidad dedidimos si el dato o no ingresa a la muestra.
- Si el dato no ingresa, la muestra no cambia. Por otro lado, si el dato si ingresa entonces elegimos al azar un dato de la muestra y lo reemplazamos por nuestro dato.

Teorema: dado un stream de n elementos la probablidad de que un elemento esté en la muestra es k/n.

Si tenemos las implementaciones antes mencionadas para calcular las métricas sobre un stream a paritr de un muestreo del del mismo.

Momento de un stream

Un stream es una colección de elementos, llamamos M_i a la cantidad de veces que el elemento i ocurrió en el stream. Se define como el momento de orden k de un stream a:

$$M^k = \sum_{i \epsilon S} M_i^k$$

El **Momento de orden 0** implica elevar a la cero. Esto es equivalente a contar 1 por cada elemento diferente del stream. Por lo tanto, el momento de orden 0 es la cantidad de elementos distintos en el stream. Calcular este momento no es sencillo porque no se puede tener en emoria todos los elementos disitntos que hemos observado y comparar.

El **Momento de orden 1** es la cantidad de elementos en total en el stream, es simplemente n. Sumamos 1 cada vez que observamos un dato en el stram a un contador.
El **Momento de orden 2** es el número sorpresa del stream. Es un indicador de si los datos del stream se distribuyen de forma pareja o de si hay un elemento que predomina sobre los demás.

Métodos para calcular el momento de orden 0:

- **Flajolet-Martin:** por cada dato que observamos en nuestro stream aplicamos la función de hashing y contamos la cantidad de 0s consecutivos en la misma a partir del bit 0. En memoria llevaremos un contador r con la cantidad máxima de ceros que hemos obervado al comienzo de cualquier resultado de la función de hashing. El momento de orden cero se lo estima como

  $$M^0(S) = 2^r$$

  Por otro lado, el algoritmo no es perfecto y es muy sensible a los valores de la función de hashing. Por ejemplo, si para algún dato la función de ahshing nos diera 0 entonces todo el algoritmo dejaría de funcionar. Una forma de solucionar esto es usando k funciones de hashing al mismo tiempo y llevando k contadores r_i en memoria. Si tenemos k estimadores el promedio no es una buena solución pq es muy sensible a los valores extremos de algunos de los estimadores, la mediana es la mejor solución, el problema es que siempre nos dará una potencia de 2.

  La solución en el algoritmo de Flajolet-Martin pasa por dividir los k eestimadores en b grupos de m estimadores cada uno y calcular como resultado final el promedio de las medias de cada grupo.

- **HyperLogLog**: es una versión mejorada del algoritmo anterior. HLL usa una única función de hashing y sobre la misma construye varios estimadores, calculando el resultado final com el promeido armónico de los estimadores. El algoritmo es así: la función de hashing nos genera un número, los primeros k bits los vamos a usar para el número de estimador. De los restantes bits contamos como siempre la cantidad de ceros al comienzo de los mismo y actualizamos el contador del estimador correspondiente si esta cantidad es mayor a la anterior. El promedio armónico se calcula de la siguiente manera:

$$M^0(S) = \frac{n}{\sum_{j=1}^{n} 2^{-R_j}}$$

  El uso del promedio armónico se debe a que este mitiga el efecto de valores muy grandes en el promedio y aumenta el efecto de valores muy pequeños.

AMS

Es un algoritmo que sirve para estimar el momento de orden 2 de un stream, es decir, su número sorpresa. Este algoritmo mantiene en memoria k estimadores, cada estimador registra un elemento del stream y la cantidad de veces que apareció el mismo. Por cada elemento del stream si el mismo está entre los estimadores simplemente aumentamos el contador en 1, si el elemento no está entre los etimadores entonces ingresa a memoria con probablidad k/n igual que en reservoir sampling si el estimador ingresa a la memoria reemplaza a otro estiamdor elegido al azar. Cada estimador estima el modelo de orden 2 del stream mediante

$$M^2(S) = n(2 * c_i - 1)$$

Siendo C_i el contador y n la cantidad de elementos del stream. Vamos a agrupar los estimadores en b grupos de m estimadores cada uno siendo el resultado final la mediana del promedio de cada grupo.

DGIM

Para el siguiente problema vamos a consierar streams binarios. Tomamos una ventana fija de m bits con m realmente muy grande y queremos calcular cuantos bits en 1 vismo en los útimos k bits con $k \leq m$. Para comprender que el algoritmo no es trivial lo que debemos entender es que m puede ser un número muy grande y necesitamos poder calcular la cantidad de unos en cualqueir ventana de tamaño k con $k \leq m$. Contar cuantos unos tenemos por cada consulta no es una opción viable.

Por lo tanto, sobre la ventana de m bits el algoritmo va a mantener k sub-ventanas que no pueden superponerse. Por cada subventana va a llevar 2 valores: la posición en la ventana donde la subventana comienza y la cantidad de unos que hay en la misma. Por cada bit procesado a la posición de cada sub ventana le sumamos 1 para mantener el offset.

Se sabe donde termina cada sub ventana porque se sabe donde comienza la anterior y la última sub-ventana está acotda por el fin de la ventana grande: anterior y la última sub-ventana está actada por el fin de la ventana grande:m. La posición donde ocmienza la ventana se calcula como módulo de m.

El algoritmo es el siguiente: procesamos bit por bit el stream, si el bit es 0 lo ignoramos completamente. Si el bit es 1 entonces creamos una ventana de tamaño 1 para ese bit con contador = 1. Luego, debemos analizar cuántas subventanas hay con 1 bit en total. Si hay 1 o 2 no hacemos nada, si hay 3 entonces combinamos la dos subventanas más viejas en una nueva sub ventana de 2 bits. ... y así sucesivamente.

Notar que el offset de las sub-ventanas siempre apunta al bit más reciente de las mismas y que ese bit siempre es 1. Recordar siempre se combinan las subventanas más viejas.

Con la estructura de sub-ventanas para calcular cuantos 1s hay en una cierta sub-vetntana de longitud k lo que hacemos es sumar la cantidad de bits de todas las subventans y la mitad de la última.

DGIM necesita en promedio log(2M) sub-ventanas y por cada sub-ventana necesita log N bits para la posición y log log N bits para la cantidad de 1s. Es decir que el espacio en memoria siempre está acotado al logaritmo del tamaño de la ventana y esto es perfectamente manejable.

El error de DGIM es aproximadamente de un 50 %, esto es bastante alto. Para reducirlo lo que hacemos es en lugar admitir 2 sub-ventanas como máximo de cada tamaño admitir r. Haciendo esto el error es de 1/r. Al admitir mayor cantidad de ventanas aumentamos el consumo de memoria.

Extensión para valores enteros

Supongamos que el strem en lugar de ser binario es de valores enteros y queremos la suma de los últimos k elemento con $k < m$ siendo m un número muy grande.

Se puede usar DGIM fácilmente para resolver este problema si suponemos que los enteros son números de z bits cada uno. Podemos entonces usar un DGIM por cada bit del número.

Luego tenemos la cantidad 1s en promedio en los últimos k elementos por cada bit del número y podemos estimar el número como: $\sum C_i * 2^i$.

Decaying windows

Es otra forma de analizar cuantos 1s aparecen en los últimos bits de un stream. Consiste en aplicar una función que vaya decayendo el peso de cada bit en 1 ya observado a medida que procesamos el stream.

Sirve para estimar para saber cuantas veces aparece cada elemento en función del tiempo. Es decir que si apareció hace mucho tiempo la cantidad es cero. Lo que sumo sale de una fórmula. También entra en juego eso de que no todos entran, tienen una probabilidad de entrar o no.

Si nos interesan los últimos n bits del stream entonces usamos una constante c = $10^{-}n$, llevamos un contador que comienza en 0 y por cada bit del stream multiplicamos el contador por 1-c y luego sumamos el bit (1 o 0). Generalmente n es un número grande.

El algoritmo funciona de la siguiente manera:

- Dividimos por la cantidad de eventos que queremos contar.
- Multiplicamos los valores en memoria por (1-c) y verificamos que no sea menor que la umbral, si tal es el caso lo quitamos.
- leemos el valor del stream y le sumamos 1. Si no está lo agregamos con valor 1.

Filtros de Bloom

Dado un stream queremos saber si los elementos que observamos en el mismo pertenecen o no a un cierto conjunto de elementos predefinidos. La base de elementos contra los cuales verificamos es siempre muy grande, no sirve mantenerlas en memoria o en un hash o estructura similar.

Un filtro de Bloom es un vector binario de B bits y k funciones de hashing 0...B. Para agregar un elemento al filtro le aplicamos las funciones de hashing y luego encendemos en 1 los bits apuntados por las funciones. Se prenden k o menos bits según haya o no colisiones.

Para verificar si un dato del stream pertenece a nuestro conjunto le aplicamos las funciones de hashing y verificamos si los bits están todos en 1 (los k bits que entran en el filtro). Si alguno está en 0 entonces el elemento no pertenece al conjunto. Podría haber un falso positivo si alguno de esos bits fue encendido por algún otro elemento. Puede haber falsos positivos pero no falsos negativos.

La probablidad de que una función encienda un cierto bit es 1/m. La probablidad de que una función no encienda cierto bit es 1-(1/m). Si tenemos k funciones de hashing la probablidad de que ninguna encienda un cierto bit es:$([1-(1/m)])^k$. Luego insertar n elementos la probabilidad de que cierto bit siga siendo cero es de:$([1-(1/m)])^{kn}$. La probablidad de que las k poisiciones a testear sean 1 entonces es: $(1-([1-(1/m)])^{kn})^k$ y una estimación para esta fórmula es: La probabilidad de un falso positivo se puede estimar por: $(1-e^{\frac{kn}{m}})^k$. Si graficamos esto obtenemos que: la probablidad de un falso positivo va bajando hasta llegar al óptimo (en función del número de funciones de hashing k) y luego sube ya que hay demasiados bits en 1 en el filtro.

Si queremos poder insertar y eliminar elementos del filtro entonces la estructura normal no sirva porque no podemos al borrar apagar los bits en 1 indicados por las k funcioens de hashing pues estos bits podrían corresponder a elementos ya insertados en el filtro y lugo al buscarlos el filtro nos diría que no están.

La solución es usar counting filter que en lugar de 1 bit por posición tienen un entreo de f bits, de estaf forma cada vez que se inserta se incrementan los enteros apuntados por las funciones de hashing y cuando se elimina se decrementa. La pertenencia es simplemente verificar que las posiciones sean distintsa de cero. La desventaja es la mayor cantidad de espacio necesaria.

Cuckoo Filters

Es una variante a los filtros de Bloom. Para responder si un objeto es miembro de un conjunto se utiliza Cuckoo hashing como estructura principal.

Diferencia entre los filtros de bloom y los cuckoo:

- Los filtros Cuckoo requiren menos espacio que los filtros de bloom.
- Los filtros Cuckoo son más rapidos para consultas.
- Los filtros Cuckoo son más lentos para hacer inserciones.
- Los filtros permiten la operación de borrado.

Almacenar las claves usando Cuckoo hashing es una opción pero entonces la estructura ocuparía una cantidad de espacio que depende del tamaño de las claves y la cantidad de claves lo cual haría que un filtro de bloom sea mucho más compacto. Por lo tanto, CF lo que hace es almacenar un hash de la misma llamado fingerprint. Estos son pequeños del orden de los 6 u 8 bits. Para almacenar estos fingerprints usaremos una tabla Cuckoo de m buckets en donde cada bucket puede almacenar b fingerprints. b determina que tanto podemos llenar la tabla antes de que las inserciones fallen.

El algoritmo funciona de la siguiente manera: La función de hashing se aplica a la clave (no al fingerprint) y nos indica en qué bucket almacenar la clave. Si el buckettiene lugar libre entonces calculamos el fingerprint de la clave, típicamente con otra fn de hashing, y lo almacenamos. Si

el bucket no tiene lugar entonces procedemos igual que el método del Cuckoo, guardamos el fingerprint de la clave indicado por f(x) y aleatoriamente removemos algún fingerprint. Paro ahroa tenemos un problema: no podemos aplicar una segunda función de hashing a la clave porque no tenemos clave! Por lo tanto, la solución es aplicar la función de hahsing al fingerprint y hacer un XOR entre la posición original y el hash del fingerprint

Limitación: como los fingerprint son pequeños la segunda posición para una clave sólo puede variar 2^f con respecto a la primera. Es decir que para la primera posición puede haber millones (dependiendo de m) de posiciones posibles pero para la segunda posición la cantidad de posiciones es limitada, es decir, que no es uniforme en m condicionada a la primera posición. Sin embargo, este esquema funciona igual de bien que el Cuckoo hashing original.

En los filtros de Bloom las inserciones nunca fallan, por lo tanto, a medida que insertamos más y más datos lo que ocurre es que la cantidad de falsos positivos aumenta. En el límite cuando todos los bits del filtro son 1 para cualquier consulta la respuesta es positiva: en los filtros Cuckoo las inserciones pueden fallar por lo que la cantidad de falsos positivos se mantiene acotada independientemente de la cantidad de datos que insertemos en el filtro (cuckoo hashing no falla siempre que no haya colecciones de 2b+1 claves que hashean al mismo bucket inicial).

Otra propiedad de los filtros de Cuckooes que soportan el borrado, si queremos eliminar una cierta clave obtenemos los dos bucket s alternativos y si encontramos el fingerprint de la calve en alguno de ellos eliminamos una copia.

Parámetros del Cuckoo:

- n: cantidad de claves.
- m: cantidad de buckets (bm = 95 % de las claves que queremos almacenar.
- b: cantidad de registros por bucket.
- f: tamaño de los fingerprints.

Probabilidad de falsos positivos de un filtro: $\frac{8\lambda}{2^f}$ siendo λ el factor de carga es decir, n/(bm).

Aumentando f podemos diminuir la cantidad de falsos positivos a costa de aumentar el tamaño de la estructura.

Count-Min

Es un algoritmo que permite para cualuquier dato de un stream estimar cuántas veces ocurrió el mismo hasta el momento.

The Heavy Hitters Problem

Es uno de los problemas que resuelve count min. Consiste en encontrar los elementos más frecuentes (populares) en un stream. Una forma de plantearlo es: dado un stream de n elementos, y un cierto valor k, siendo n un número muy grande y k un número relativamente chico encontar los elementos que ocurren al menos n/k veces en el stream.

Teorema: no hay forma de resolver el problema de los Heavy Hitters en ninguna de sus variantes en O(n) sin que la complejidad especial se dispare. Por lo tanto, necesitaremos una solución aproximada ya que la solución exacta es **imposible.**

El problema de HH aproximado

En el problema aproximado tenemos un strem S de n elementos y dado un cierto k queremos cumplir con dos condiciones:

- Todo valor que ocurre al menos n/k veces en S está en el resultado.
- Todo valor en el resultado ocurre al menos n/k -en veces en S.
- El espacio debe ser O(1/e).

Cuando e tiende a cero el problema se vuelve el esacto y el costo tiene a infinito.

0.0.1. The Count-Min Sketch

Es equivalente a tener l counting filters. Necsitamos entonces l funciones de hashing, llamaremos b al espacio d direcciones de las mismas. Por cada dato del stream, aplicamos las l funciones de hashing e incrementamos el bucket apuntado por las funciones en cada uno de los filtros. La estimación del algoritmo sobre la cantidad de veces que apareció un ítem es el mínimo de los l contadores. El problema de los top k elementos se resuelve con una tabla de k elementos y cada vez que ocurre un elemento en el stream etimamos su cardinalidad y actualizamos el ranking según sea necesario.

Count-Min nunca va a estimar un número menor a la cardinalidad real de un elemento, pero puede estimar un número mayor debido a las colisiones.

PageRank y sus derivados

En un principio los motores dedicados a buscar páginas web trabajaban analizando pura y exclusivamente el contenido de etas páginas. De acuerdo a la consulta del usuario se asignaba un puntaje a cada página que dependía de factores relacionados a los términos buscados.

Esto no funcionó porque se podía generar spam muy facilmente entonces ahí es cuando surge el algoritmo de PageRank que busca ranquear los resultados de la búsqueda de forma independiente del contenido de las mismas. Se basa en la estructura de links que existe entre documentos/páginas web. El concepto básico es que cada página tiene una cierta **importancia** que es intrínseca y depende de los links que lleven a dicha página. Cuantos más links nos puedan llevar a una cierta página más importante será la misma.

Para entender la idea de PageRank debemos entender el modelo matemático detras de PageRank que es el modelo de Random surfers.

Un Random surfer es un navegante aleatroio en la web. Este navegante comienza en cualquier página al azar de tdas las disponibles. Desde esa página nuestro navegante elige un link al azar de la página de la página en la que se encuentra y naveaga a la página seleccionada, repitiendo este proceso en la nueva página que ha visitado. Este proceso se repite indefinidamente en las sucesivas iteracioens del navegante aleatorio.

El concepto detrás de pagerank consiste en darle a cada página un peso que es igual a la probablidad de que nuestro RandoSurefer termine su recorrido en esa página luego de n saltos.

El modelo del Random Surfer es un modelo de Markov en el que tenemos nodos como páginas y aristas como links. La probablidad de cada arista es 1 sobre el total de links en la página. Cada arista tiene probabilidad inicial 1 / # nodos. Por lo tanto, luego de un salto las probablidades de cada página se pueden calcular ocmo el producto de la matriz de Markov por las probablidades

iniciales (el vector obtenido puede meterse de nuevo en una multiplicación y obtener un nuevo resultado).

El PageRank se define como las probablidades luego de infinitos saltos. Esto nos lleva a pensar si PageRank converge y si converge a qué valor lo hace.

Convergencia

Teorema de distribución estacionaria: Si A es una matriz estocástica entonces existe v tal que: Av= v.

Una matriz estocástica es aquella en la cual todas sus filas o columnas suman 1. En el caso de PageRank por definición todas las columnas suman 1 y por lo tanto la matriz es, en principio, estocástica.

Teorema de Perron-Frobenius: El autovalor principal de una matriz estocástica es igual a 1. No puede existir un auto valor mayor.

Por lo tanto, una matriz estocástica tiene un único autovalor igual a 1 y es el autovalor principal de la matriz.

Power Method: sea A una matriz diagonalizable y sea x_0 un vector aleatorio si repetimos $x_0 = Ax_0$ en el límite x_0 es el autovector asociado al autovalor de mayor valor absoluto de la matriz. Esto sirve apra cualquier matriz.

En conclusión la matríz de PageRank tiene un único autovalor principal igual a 1 y el método iterativo converge a un autovector asociado a dicho autovalor.

Teorema fundamental de las Cadenas de Markov: para un Random Walk lo suficientemente largo en una cadena de Markov la probablidad de terminar en un cierto vértice v es independiente del vértice en el cual se inició el Random Walk.

Mediante un Random Walk lo suficientemente largo podemos calcular la probablidad de terminar en cada una de las páginas de nuestro grafo y al ser esto independiente de donde comenzamos nos sirve como una medida de importancia de ellas.

PageRank en cualquiera de sus variantes es un algoritmo que busca el autovector principal de una matriz estocástica. Si entendemos esto, ya entendimos todo el algoritmo.

Por lo tanto, existen casos patológicos que suelen preentarse en la web que hacen que tengamos que realizar algunas variantes en el algoritmo para que las condicioens planteadas sigan siendo válidas.

Dead Ends

Dead End es una página desde la cual no se puede ir a ninguna otra página. En esta situación, nuestro navegante aleatorio, eventualmente va a caer en el dead end y quedará atrapado en esa página. Por lo tanto el dead end va a capturar todo el PageRank y las otras páginas van a quedar en cero. El problema se hace evidente al analizar la matriz y ver que la misma no es estocástica, dado que la columna correspondiente al dead end se encuentra llena de 0.

para que un PR funciones la matriz siempre tiene que ser estocástica. cuando una página es un dead end tenemos que corregirla. Para ello agregamos links a todas las otras páginas

con probablidad 1/n, indicándole al navegante aleatorio que puede visitar cualquiera de las diponibles desde allí.

Siempre que encontremos dead ends el primer paso es corregirlos para asegurar que la matriz sea estocástica.

Spider Traps y Teletransportación

Supongamos que entre dos nodos D y E hay una arista que nace en D y muere en E y una que nace en E y muere en D. Ninguno de los nodos tiene aristas a otra arista del grafo.

Podemos ver en el grafo que una vez que nuestro navegante aleatorio llege a D va a quedar atrapado en D y E si poder salir. Por lo tanto, el Pagerank va a distirburse entre D y E y el resto de las páginas va a quedar en cero. Esto es lo que llamamos un Spider Trap.

Si observáramos la matríz asociada al grafo veremos que es estocástica por lo que en principio no hace falta hacerle ninguna corrección (por eso es tan difícil detectar estas trampas). La solución a este problema fue el gran descubrimiento dle algoritmo de Page-Rank y se llama teletransportación.

Cada vez que nuestro navegante aleatorio llegue a una página vamos a darle dos posibilidad: con probablidad B le vamos hacer elegir un link al azar y navegarlo como vinimos trabajando hasta el momento y con probablidad 1-B vamos a teletrasportarlo a cualquier otra página al azar (siendo todas equiprobables).

La forma de aplicar teletrasportación es la siguiente: $A = B(A_{original})+(1-B)(A_{completa\ de\ 1/n})$

Observemos que el proceso de multiplicar la matriz A por el PageRank es equivalente a hacer: $V_p = B(A_{original})V_p + (1-B)(V_{completo\ de\ 1/n})$

La teletransportación soluciona otro problema que es el caso de los grafos periódicos. Si el grafo es periódico el PageRank no converge sino que oscila. Para que esto no ocurra debe ser posible garantizar que luego de un Random Walk lo suficientemente largo existe una probablidad positiva de estar en cualquier vértice del grafo en el próximo paso.

Una forma de asegurar esto es que se usa en Cadenas de Markov es agregando un link en cada nodo a si mismo, pero en PageRank no es necesario hacer esto porque la teletransportación asegura la periocidad del grafo: en cualquier momento con una probablidad positiva podemos teletransportarnos a cualquier otro nodo.

Interpretado de B

El parametro B es la probablidad de seguir un lin es decir de no teletrasportarse. En general se usa B = 0.8 o similar.

Analicemos el efecto de B. Si B =1 tenemos el caso sin teletransportación. En este caso hay un únco random walk sobre nuestra cadena de Markov. Si B¡1 entonces la cadena se reinicia de vez en tanto y cuanto menor sea B más random walks vamos at ener y más cortos van a ser estos. En el límite B tendiendo a 0 tenemos infinitos random walks de longitud 0 es decir que simplemente tomamos nodos al azar de nuestro grafo para visitarlos y el Rank de cada 1 sera 1/N.

Con el agregado de la teletransportación y recordando que es necesario arreglar los dead-ends tenemos desarrolaldo el algoritmo de PageRank de forma completa.

Page Rank en la práctica

Tenemos una matriz con billones de nodos entonces no podemos pensar en calcular el producto de lamatriz por el vector de PageRank de las páginas ya que incluso para una sóla iteración este proceso sería inviable. Por suerte existe una relación simbiótica entre PageRank y MapReduce.

Considermos de qué forma queremos procesar el pagerank. Podemos ir página por página y en cada página tenemos una serie de links a otras páginas, sabemos además que el pageRank de cada página. Entonces estando en una página sabemos que esta transfiere su PageRank a cada una de las páginas linkeadas de forma equiprobable.

Podemos entonces simplemente sumar a cada una de las otras páginas el PageRank correspondiente a la página origen dividido por la cantidad de links y multiplicando por B. Una vez que hisimos esto para todas las páginas simplemente sumamos al vector resultado la teletransportación que es (1-B)(1/N). Esto nos da un algoritmo muy simple para procesar el pagerank si tenemos el vector de PageRan qen memoria.

Observemos que la matriz de links en realidad no hace falta. Sólo necesitamos el vector con el PageRank de cada página y por cada página los links que tiene la misma.

Este método es suficiente para grafos realmente grandes pero no para billones de páginas, entonces recurrimos a Map Reduce.

PageRank via Map Reduce

El proceso de Map procesa una ptágina con sus links y emite un registro por cada link en la ptágina transfireiendole la parte del pageRank que le corresponde. El proceso Reduce suma todos los puntajes obtenidos y agrega la teletransportación.

Este proceso se repite tantas veces como sea neceario para llegar a la convergencia. Vamos alterando fases de Map y Reduce en donde cada reduce actualiza el valor del PageRank para cada página y sirve como dato para la siguiente fase de Map.

Si agregamos nuevas páginas podemos incorporarlas simplemente con PR = 0 y tarde otemprano el algoritmo va a converger, no es necesario volver a empezar el algoritmo ya que comenzando con cualquier vector el método iterativo siempre converge al autovector asociado al autovalor principal de la matriz.

TopicRank

En el algoritmo tradicional nos teletransportamos con igual probablidad a cualquier página, pero esto no lo podemos cambiar. Si decidimos, por ejemplo, teletransportarnos sólo a aquellas páginas que tratan sobre un determinado tema tenemos una variante de PageRank llamada TopicRank en donde las búsquedas sobre un cierto tema van a darnos como resultado páginas que tratan sobre dicho tema. Esto se logra manipulando el concepto de teletransportación.

Supongamos que sabemos, de alguna forma, cuáles son las páginas que tratan sobre un dterminado tema, necesitamos correr una versión de PageRank por cada tema. para poder realizar el PageRank temático en lugar de sumar (1-B)/N a cada página sumamos (1-B)/$|C|$ en donde $|C|$ es la cardinalidad del tema, es decir, la cantidad de páginas que están calificadas dentro del tema que nos interesa.

TrustRank

Es una variante de PageRank para lograr disminuir el efecto de las spamfarms. Spammers crean una gran cantidad de páginas que linkean a la página en cuestión y links desde la páginas a estas otras, formando una red. Luego insertan links en páginas públicas que llevan a la página objetivo.

La forma en la que TrustRank combate los SpamFarms es, nuevamente, mediante la manipulación de la teletransportación. La idea es simple: sólo nos podemos teletransportar a páginas confiables. Esto es equivalente a tener un TopicRank en donde el topic es que la página es confiable.

Determinar qué paginas son confiables o no no es una tarea sencilla, podemos usar dominios confiables y el resto recurrir a la clasificación manual.

Podemos determinar la masa del spam de una página como: Spam mas $= \frac{PR-TR}{PR}$

A partir de la masa de spam podemos establecer un cierto umbral y eliminar las páginas que estén por encima del mismo o eliminarlas manualmente mediante una supervisación humana.

SimRank

Con las variantes posibles para la teletransportación tenemos el caso en el cual sólo podemos teletansportarnos a una única página. Lo que obtenemos entonces es una serie de random walks en los cuales siempre volvemos al mismo punto de partida. El pageRank de cada página representa entonces la probablidad de llegar a cada página desde una cierta página origen y a esto se lo llama simrank.

El proceso será igual solo que modificamos el vector que acompaña a (1-B) por uno que tenga un uno en el vector que queremos calcularle el simrank. El pagerank de ese vector no nos interesa, lo que nos interesa es la página más similar a ese vector.

SimRank via Montecarlo

El problema del algoritmo SimRank es que no podemos calcular el SimRank de cada página. La solución pasa por no calcular el SimRank sino por simularlo mediante el método de Montecarlo. Cuando queremos obtener las páginas más similares a alguna hacemos una simulación de random walks comenzando en la página. A medida que realizamos los RW registramos por que páginas pasamos y cuántas veces visitamos cada una. Luego de simular una buena cantidad de RW, lo cual puede hacerse muy rápidamente, tenemos una muy buena aproximación del SimRank de la página simplemente listando las demás en orden decreciente por la cantidad de visitas.

Panther

Es un algoritmo muy similar a la versión MonteCarlo de SimRank que tiene propiedades muy diferentes. En este algoritmo creamos RW al azar pero a partir de cualquier nodo, no sólo a partir del nodo que nos interesa. La semejanza entre nodos se define como el porcentaje de caminos en los cuales los nodos aparecen juntos.

VisualRank

Es un algoritmo para rankear imágenes en un buscador de imágenes. Lo que podemos hacer es buscar imágenes en las cuales aparezca el término buscado, ya sea en su título o en algún metadato.

El problema que surge es que no tenemos idea cual es la más relevante. Para resolver esto lo que hace Visual Rank es que la imágen más relevante será aquella que más se parece a todas las demás.

Visual Rank comienza armando un grafo entre todas las imágenes recuperadas, un grafo completo en donde cada arista representa la semejanza entre las imágenes. Luego corriendo PageRank en el grafo, usando la semejanza entre las imágenes como la probablidad de visitar una luego de otra, podemos rankear las imágenes y presentar el resultado final. El nuevo problema es determinar la semejanza entre imágenes.

Para calcular la semejanza entre dos imágenes VisualRank hace uso del algoritmo de SIFT (Scale Invariant Feature Transform). Este algoritmo extrae de cad aimagen un conjunto de puntos que no se modifican con diferentes escalas, rotaciones o iluminación.

Cada punto extraído por SIFT tiene un descriptor de 128 elementos (un vector). La semejanza entre dos imágenes se calcula mediante la distancia entre todos los puntos SIFT de una imágen contra todos los puntos SIFT de la otra.

Comapara todos los puntos SIFT contra todos por cada par de imágenes es una tarea muy costosa por lo que VisualRank usa LSH. La semejanza será la cantidad de colisiones que tuvimos en total dividiendo por el total de puntos SIFT.

TextRank

Consiste en aplicar Page Rank sobre textos. Este es uno de los algoritmos más efectivos para realizar resúmenes automáticos. Este último consiste en extraer de un texto sus frases más significativas y presentarlas luego en orden de aparición.

Para extraer las frases más significativas TextRank arma un grafo con todas las frases del texto y usa alguna métrica de distancia entre frases para calcular la distancia entre las mismas (TF-IDF, BM25).

Con el grafo construido TextRank simplemente corre PageRank sobre el mismo y luego extrae las k frases de mayor PageRank para realizar el resumen.

Si en lugar de frases extraemos palabras de un texto, tenemos un algoritmo para extraer los KeyWords más importantes del texto. Para la semejanza entre palabras podemos usar la cantidad de veces que las palabras ocurren o la distancia promedio entre las palabras cuando ocurren en la misma frase.

HITS

Es un algortimo para calcular la importancia de los nodos de un grafo.

La idea de HITS es calcular dos puntajes para cada páginas:

- HUB: es una página que linkea a bunas páginas.
- Autoridad: es una página que es linkeada por buenos HUBS. Es similar al PageRank.

Inicialmente cada página tiene valor 1 como HUB y valor 1 ocmo Authority. Luego el algoritmo comienza a iterar. EL valor de Authority de una página es la suma del valor HUB de todas las páginas que linkean hacia ella. Una vez que calculamos el valor de Authority de cada página calculamos el valor de HUB de cada página como la suma de los valores de Authority de todas las páginas hacia las cuales la página linkea. Los alores de HUB y Authority se normalizan para que

todos sumen 1. Este proceso se repite hasta alcanzar la convergencia.

Teorema: HITS normalizado en cada paso converge a un único valor tanto de HUB como de Authority.

Llamamos A a la matriz de adyacencias del grafo, la matriz de links. Indicamos con 0s y 1s si existe o no un link desde una página hacia otra.

Llamamos h al vector de tamaño N con los valores como HUB de capada página y llamaremos a al vector con los puntajes como Authority de cada página. Tenemos que:

1. $h = Aa$
2. $a = A^t h$
3. Reemplazando 1 en 2: $h = AA^t h$.
4. Reemplazando 2 en 1: $a = A^t Aa$

Es decir que h converge al autovector asosciado al autovalor principal de AA^T y a converge al autovector asociado al autovalor principal de $A^t A$ demostrando el teorema de la convergencia de HITS.

La forma correcta de normalizar HITS para garantizar la convergencia es dividiendo cada puntaje por la raiz cuadrada de la sumatoria del cuadarado d todos los puntajes, es decir, obligando a que los vectores h y a tengan una norma igual a 1.

Redes sociales

Una red social es un grafo en el cual los nodos son usuarios y las aristas expresan las relaciones entre los mismos.

Propiedades de los grafos

Algunas definiciones:

- N: Cantidad de nodos del grafo.
- E: cantidad de aristas.
- k_i grado de cada nodo.
- k grado promedio del grafo. Si el grafo es no dirigido entonces k = 2E/N.
- Grafo completo: aquel que tiene todas sus aristas. Si el grafo es no digido y completo entonces k = N-1, E = N(N-1) /2. Las redes sociales no son grafos completos sino que son grafos dispersos. Para la mayoría de las redes sosciales el grado promedio es un número muy pqueño en relación a la cantidad de nodos en la red social.

Almacenamiento de Grafo

La forma más evidiente de almacenar el grafo de una Red Social es mediante una lista de adyacencias en donde por cada nodo se almacenan cuáles son sus nodos vecinos. Esto es porque k generalmente es un número muy pequeño y por lo tanto cada nodo tiene pocos vecinos.

Incluso para grafos muy extensos es posible tener el grafo completo de la red social en memoria, esto es muy importante ya que en muchos clasos no se analiza correctamente el poder de representar los grafos de forma completa.

Componentes conexos

En una red social en general existe un único componenete conexo gigante. Esta es una propiedad que tienen todos los grafos con muchos nodos. De esto se desprende que las redes sociales son, casi siempre, grafos conexos.

Diametro

El diametro de un grafo es el máximo de todos los caminos mínimos. En otras palabras, cuál es el máximo número de aristas que tenemos que visitar para llegar un nodo del grafo a otro. Las redes sociales tienen un diámetro muy bajo, lo caul da origen al fenómeno del mundo pequeño. Otra propiedad de las redes sociales es que a medida que las redes sociales crecen el diámetro en lugar de aumentar disminuye.

Coeficiente de Clustering

El coeficiente de clustering mide que tan conectados están los vecinos de un nodo. Sea k_i el grado del nodo entonces sus k_i vecinos pueden tener a lo sumo $1/2k_i(k_i - 1)$ aristas. Llamando a e_i a la cantidad de aristas entre los vecinos de i, el coeficiente de lcusterin para el nodo i es:

$$c_i = \frac{2e_i}{k_i(k-1)}$$

El coeficiente de clustering promedio de la red se define como:

$$C = \frac{\sum_{i=1} c_i}{N}$$

En una red social el coeficiente de clustering es en genearal un número alto. Los vecinos de un nodo tienden a estar conectados entre sí precisamente por tener en común su relación con ese nodo. A este fenómeno se lo conoce como clausura traingular.

Distribución del grado

La distribución del grado estudia de qué forma se distribuye el grado de los nodos de una red social. un fenómeno muy curioso es que en casi tododas las redes sociales la distribución se distribuye d eacuerdo a una ley de potencias (power law). En este tipo de distribuciones hay muchos nodos con grado bajo y pocos nodos con un grado muy alto pero estos dominan la distribución por el fenóneno conocido como long tail.

Modelos de grafos

Resumen de las características de los grafos en redes sociales:

- **Diámetro**: muy pequeño.
- **Coeficiente de clustering**: muy alto.
- **Componentes conexos**: 1.
- **Distribución del grado**: Power Law.
- **Camino mínimo promedio**: bajo.

Modelo de Erdös-Renyi

Es un grafo aleatorio con N nodos y probablidad p para cada arista. A estos grafos se los llamará G_{np}. Para generar este grafo lo que se puede hacer es calcular la cantidad de aristas en el grafo como pN(N-1)/2 y luego formar cada una de estas aristas tomando dos nodos al azar.

Diámetro: el diametro de G_{np} está en el orden de O(log n). Es como un estimador de el díametro de un grafo. Lo que sucede es que el diamtero de una red social no necesariamente obedece a características de la misma sino que puede ser simplemente, una propiedad de un grafo aleatorio del mismo tamaño.

Distribución del grado: el grado promedio de un nodo en G_{np} es (n-1)p. Siendo p la probablidad de que cada arista exista. La distribución del grado de G_{np} es una distribución binomial. Esta distribución tiene forma normal y no se parece en absoluto a la distribución de una ley de potencias que es la que esperamoos en una red social, por lo tanto, **se puede decir que la formación de aristas de una red social no es aleatoria puesto que si lo fuera la distribución no sería normal.**

Coeficiente de clustering: como el coeficiente de clustering era: $c_i = \frac{2e_i}{k_i(k_i-1)}$ se puede calcular: $e_i = p\frac{k_i(k_i-1)}{2}$ y entonces: $C_i = p = \frac{p}{n-1} \approx \frac{k}{N}$ p es un número muy pequeño pero esto no sucede, en una red social el grado de conexión entre los amigos de un determinado nodo es realmente altísimo con respecto a lo que ocurriría simplemente por azar.

Del estudio de G_{np} se puede concluir que una red social no es random, el coeficiente de clustering denota una estrucctura local mucho más alta que la que se esperaría por simple azar y la distribución del grado es completamente diferente.

Modelo de Barabasi-Albert

No genera las aristas del grafo de forma aleatoria sino que utiliza un modelo llamado preferential attachment donde la idea es que la probablidad de agregar una arista en el grafo depende del grado de cada nodo. Cuanto mayor sea el grado del nodo más probable es que genere una nueva arista, este fenómeno se llama: *the rich get richer.* La probablidad de generar contactos será:

$$p_i = \frac{k_i}{\sum k_j}$$

El uso de preferential attachmente genera una distribución que es una ley de potencias con exponente $\alpha 4$. El diámetro en este modelo es O(loglogN) que también es compatible con el diámetro de una red social.

Modelo de Jackson-Rodgers

Es una combinación de G_{np} y el modelo de Barabasi-Albert. Se usa un coeficiente α que será la proablidad de generar un link al azar en la red. Luego 1-α es la probablidad de generar un link siguiendo el proceso de preferential attachment.

Es decir que si $\alpha = 0$ es igual a preferential attachmente y si es igual a 1 es el modelo de G_{np}.

Empíricamente se puede demostrar que la mayoría de las rede sociales puede modelarse la formación de links de forma muy precisa para un cierto parámetro α que depende de la red. El problema del modelo es la necesidad de encontrar el valor de α que se tiene que buscar mediante una regresión lineal.

El mundo pequeño

En un grafo aleatorio G_{np} se tiene presente el fenómno del mundo pequeño pero el coeficiente de clustering promedio es muy bajo. SI se organizan los nodos en una grilla en donde cad anodo se conecta con k vecinos se tendrá un coeficiente de clustering alto pero no ocurrirá el fenómeno dle mundo pequeño. El problema es cómo encontrar un modelo que nos de ambas cosas.

El modelo de Watts Strogratz

En este modelo se parte de una grilla regular, por ejemplo, en donde todo nodo está conectado a sus dos vecinos más cercanos. Esta grilla tiene clustering alto pero también diámetro grande, es decir que no responde al fenómeno del mundo pequeño. Por cada arista con probablidad q se va a cambiar uno de sus nodos por cualquier otro del grafo.

Si se usa q = 0 se obtiene la grilla regular con alto clustering y alto diámetro. Si se usa q = 1 se obtendrá G_{np} con bajo clustering y bajo diámetro. El objetivo es encontrar q tal que obtenga alto clustering y bajo diámetro como en las Redes sociales. Se puede demostrar que esto último se puede obtener con q bajos, es decir, que con agregar unos links al azar hace que el modelo de Watts Strogratz se convierta en una red social.

Lamentablemente no tiene la distribución de grados que esperamos de una Red Social.

La clausura triangular

La clausura triangular es el mecanismo que origina el coeficiente de clustring alto en las redes sociales y se enuncia así: si A conoce a B y c entonces tarde o temprano B conoce a C.

La existencia de una cantidad inusualmente alta de triángulos automáticamente implica un coeficiente de clustering alto, contra tirángulos dentro de un grafo, es entonces una de las formas más efecitvas de determinar si el grafo responde a las características de una red social.

Contando triángulos de un grafo

Método trivial: consiste en tomar cada nodo y por cada nodo tomar a su vecinos y verificar si para cada par de vecinos existe un link entre ellos. Este método es ineficiente e inviable en grafos grandes.

La matriz de adyacencias de un grafo es M. Si se calcula M^2 en el elemento i,j representa los caminos de longitud 2 que empiezan en i y mueren en j. La diagonal de la matriz es el grado de cada vértice. De la misma manera, M^3representa los caminos de longitud 3. La diagonal de matriz representa ahora todo los caminos de longitud 3 que empiezan y terminan en el mismo nodo. Por lo tanto la diagonal dividido 2 es la cantidad de triángulos en la que participa cada nodo. Esto quiere decir que la traza de la matriz dividido por 6 es la cantidad de triángulos del grafo (dividimos porque cada triángulo tiene 3 nodos participantes).

Pero no hace falta elevar la matriz al cubo ya que sabemos que si λ es autovalor de M, λ^n es autovalor de M^n y se sabe que la suma de los autovalores es igual a la traza de la matriz por lo tanto se puede estimar la cantidad de triángulos en el grafo como:

$$\frac{1}{6}\sum_{i=1}^{n}\lambda_i^3$$

Los autovalores de la matriz se pueden obtener de forma eficiente con el método de Lanezos y ni siquiera hacen falta todos porque se puede estimar la cantidad de triángulos a partir de los k

autovalores de mayor valor absoluto de la matriz con un error muy pequeño.

El método de Lanezos está basado en el power method con el que se acalcula PageRank. Como un múltiplo de un autovector es autovector se va a normalizar el vector resultado de cada iteración del power method de forma tal que tenga norma 1. De esta forma se puede obtener el autovector asociado al autovalor principal de norma 1. el autovalor asociado se puede calcular como: $\lambda = x^t Ax$. donde x es el autovalor principal.

Una vez que se obtiene el autovector asociado al autovalor principal se puede neutralizar el efecto del mismo en la matriz para enocntrar el segundo autovector. Para eso se usa lo siguiente: $A* = A - \lambda x x^t$. Aplicando el power method a la nueva matriz A* se obteiene el sagundo autovector. Repitiendo este método se obtienen autovalores y autovectores de forma iterativa.

Triángulos locales

Los triángulos lcoales son la cantidad de triángulos en los cuales participa cierto nodo. Esto nos da una medida de que tan sociable es el nodo en la Red. Para obtener esto hacemos:

$$\nabla_i = \frac{\sum_j \lambda_j^3 u_{ij}^2}{2}$$

en donde u_{ij} es la ísima entrada del jesimo autovector.

Si se grafica la cantidad de tirángulos en los que participa un nodo lo que tenemos es una ley de potencias.

Centralidad

Es una medida de la importancia que cada nodo tiene en la red.

Grado

Es la medida más simple de centralidad, a mayor vecinos más importante el nodo. En esta métrica se supone que no importa quienes son los vecinos de un nodo sino simplemente cuántos son. Para poder comparar la centralidad de dos nodos (de una misma red o no) es recomendable normalizar el grado dividiendo por el grado máximo que sabemos que es N-1.

El grado no captura la importancia que tiene un nodo como nexo de dos o más comunidades (brokerage). Para medir esto se usa el concepto de Betweenness.

Betweenness

Se interpreta como la cantidad de caminos mínimos que pasan por un nodo o arista. Es decir que puede usarse tanto para calcular la importancia de los nodos como de aristas. La definición formal es:

$$C_B(i) = \sum_{j<k} G_{jk}(i)/G_{jk}$$

En donde G_{jk} es la cantidad de caminos mínimos que conectan j y k, y $G_{jk}(i)$ es la cantidad de caminos mínimos entre j y k por los que participa i. En general se puede normalizar el betwenness dividiendo por (n-1)(n-2)/2 que es la cantidad de pares de vértices que existen excluyendo al vértice mismo. La normalización es opcional.

En algunos casos no se quiere medir la cantidad de amigos que tiene un nodo ni que tan .en medio"se encuentra el nodo sino que se quiere medir que tan cerca está el nodo del centro de la red. Esto se puede medir mediante la centralidad que se llamará closeness.

Closeness

El concepto de closeness está basado en el promedio de los caminos mínimos entre el nodo y todos los demás nodos de la red. Como este valor es más chico cuanto más central es el nodo se usará el recíproco, es decir, 1 dividido el promedio de los caminos mínimos. De esta forma la centralidad dará mayor cuanto más cerca del centro de la red se encuentre el nodo.

$$C_c(i) = [\sum_{j=1}^{N} d(i,j)]^{-}1$$

opcionalmente se puede normalizar dividiendo por (N-1).

Eigenvector centrality y PageRank

El concepto de Eigenvector centrality es que la centralidad de un nodo depende de la centralidad de los nodos que apuntan hacia dicho nodo. Es decir, Eigenvector Centrality es la misma idea que PageRank sin teletransportación.

Centralidad de Bonacich

La centralidad de Bonacich es una variante de la EigenVector centrality. Se define de la siguiente forma:

$$c_i(\beta) = \sum_j (\alpha + \beta c_j) A_{ji}$$

$$c(\beta) = \alpha(I - \beta A)^{-1} A1$$

donde α es una constante de normalización. β determina qué tan importante es la centralidad de los vecinos. A_{ij} es la matriz d eadyacencias. Es decir que solamente se suman $\alpha + \beta c_j$ para aquellos nodos j que llevan a i.

Cuando β es un número chico sólo los vecinos inmediatos son importantes. Cuando β es un número grande el efecto global de la red es el factor más importante. Notar que si $\beta = 0$ entonces queda ci = $\sum_j \alpha A_{ij}$ que es la definición de grado del nodo.

Cuando β ¿0 los nodos tiene mayor centralidad cuando tiene aristas hacia otros nodos con mucha centralidad. Cuando $\beta < 0$ los nodos tiene mayor centralidad cuando tienen aristas hacia otros nodos con poca centralidad.

Centralidad de la red (Freeman

Se usa para medir el grado de centralidad de la Red en su totalidad.

$$CD = \sum_{i=1}^{g} \frac{[C_D(n*) - C_D(i)]}{(N-1)(N-2)}$$

en donde $C_D(i)$ es la centralidad del nodo i de acuerdo a alguna métrica y $C_D(n*)$ es la centralidad máxima de cualquier nodo de la red.

La centralidad de Freeman es un número entre 0 y 1. Una red con centralidad 1 indica que hay un nodo que es el que tiene máxima centralidad y todos los demás son mínimos, es decir una topología tipo estrella. Una centralidad cercana a 0 indica que una red cuya topología está fuertemente des-centralizada y todos los nodos tienen centralidades muy similares. En general esto evidencia una estructura más caótica.

Detección de Comunidades

Uniones fuertes y débiles

Se pueden caracterizar las aristas de una red según la fuerza de sus uniones. En general una unión es débil cuando al removerla el camino mínimo entre los nodos que la arista unía se incrementa. Se define como **span** de una arista como la distancia entre los nodos si se remueve la arista. Un *puentelocal* es una arista cuyos vecinos no tienen vecinos en común.

Las uniones débiles son muy importantes en las redes ya que son aquellas que permiten descubir cosas. La conclusión es que **una red social es un conuunto de comunidades unidas entre sí por uniones débiles. Detectar estas comunidades es muy importante para el estudio de las redes sociales.**

Modularidad

La función modularidad permite evaluar qué tan modular es una red separada en un ciertto conunto de comunidades. La función supone la existencia de una función $\delta(v1, v2)$ que vale 1 si los vértices v1 y v2 están el la misma comunidad y 0 en otro caso.

La definición de modularidad es la siguiente:

$$Q = \frac{1}{2m} \sum_{vw} [A_{vw} - \frac{K_v K_w}{wm}] \delta(c_v, c_w)$$

Donde A_{vw} es la matriz de adyacencias. K_v y K_w son el grado de los vértices, m es la cantidad total de aristas en el grafo y δ es la función que ya se ha explicado anteriormente. Para redes completamente aleatorias Q = 0.

Algoritmo de Louvain

El algoritmo es el siguiente: inicialmente cada nodo pertenece a su propia comunidad, es decir, que hay tantas comunidades como nodos y la propia función delta valo 0 para todos lo pares de nodos. La modularidad inicial es 1/(2m). Luego por cada nodo de la red se busca signarlo a la comunidad vecina que maximice la modularidad. Una vez que se han recorrido todos lo nodos asinando a cada uno la comunidad que corresponde se considerará a cada comunidad como si fuera un nodo y se repite el proceso. Cuando se considera a cada comunidad un nuevo nodo los links entre nodos dentro de esta comunidad se consideran links del nodo a si mísmo.

En redes que soy muy grandes el algoritmo de Louvain tiende a detectar grades comunidades ya que suele pasar que mergear dos comunidaes aumente la modularidad, es decir, que la resolución del algoritmo depende del tamaó de la red y en redes muy grandes el algoritmo no puede detectar comunidades pequeñas.

Algoritmo de Girwan-Newmann

El algoritmo realiza una descomposición de la red usando el cálculo de betwenness para cada arista. En cada paso el algoritmo elimina la arista con mayor beteweenness. El algoritmo se detiene cuando se ha llegado a la cantidad de comunidades que estamos buscando o con algún otro criterio.

Clustering Espectral

El algoritmo de clustering espectral puede aplicarse directamente sobre la matriz de adyacencias de la red de forma tal de obtener una cantidad de comunidades que tenemos que conocer con anticipación.

Infomap

NO LO ENTENDÍ

Cascadas en Redes Sociales

Se estudiará el flujo de la información en las redes sociales. En concreto, se estudiará la forma en la cual la información se difunde a través de la topología de la red.

La difusión de la información se inicia cuando un usuario de la red genera el primer mensaje sobre dicho tema, este mensaje se va propagando por la topología de la red y llega alcanzando un cierto número de nodos (usuarios). AL conjunto de nodos a los cuales llega la información se lo llamará una **cascada**.

Para poder estudiar las cascadas en las redes sociales usaremos el siguiente modelo de difusión que es: si al menos a q % de mis vecinos difunden algo yo también lo hago. Esto definirá una cascada q.

Es de particular interés estudiar cuándo se detiene una cascada en una red social y cómo maximizar la influencia, es decir, cómo llegar a la mayor cantidad de nodos posibles.
Los custers son obstaculos para la difusión de la información en las redes sociales.

Por otro lado, maximizar la influencia implica llegar a la mayor cantidad de nodos posibles en una red mediante el uso de la menor cantidad de nodos posibles. Es decir, que interesa descubrir un conjunto pequeño de nodos influyentes de forma tal que la información llegue a la mayor cantidad de usuarios posibles.

Se trata del conocido problema de encontar el conjunto mínimo cde vértices dor forma tal que etos cubran todas las aristas del grafo. Se quiere encontrar al conjunto mínimo de vértices de forma tal que estos cubran todas las aristas del grafo. Es necesario entnces aproximar el problema. Para ello, se define a f como la función que recibe un conjunto de nodos y devuelve la influencia de los mismo. Esta función es una función submodular y esto quiere decir que si se agreaga un nodo a un conjunto el resultado dará menor o igual ganancia que agragarlo a un subconjunto.

Cuando f es una función monótona y submodular un algoritmo tipo greedy es en el peor caso un 63 % de la solucón óptima. Esto es bastante aceptable por lo que se puede usar un algoritmo greedy muy simple para maximizar la influencia: en cada paso se agrega el nodo de mayor ganancia de influencia nos aporte.

El problema es que la función f que mide la influencia de un nodo en realidad no existe. Afortunadamente esta función se pude simular mediante un modelo de cascadas y la topología de la red se podrá estudiar a cuantos nodos llega la información a partori de un cierto conjunto de nodos, agregando siempre en cada paso el nodo de mayor ganancia de influencia da.

En definitiva se trata de resolver un problema mediante un algoritmo greedy que optimiza una función que en realidad no existe.

Redes sociales con signo

En ciertas redes sociales las relaciones entre los usuarios pueden expresarse de forma positiva o negativa. Anlizaremos un algoritmo para predecir el signo de una relación que aún no existe entre dos usuarios.

Triángulos que son naturalmente establees (balanceados) y son los que obedecen las siguientes reglas:

1. Los amigos de mis amigos son mis amigos.
2. los enemigos de mis enemigos son mis amigos.

Existen relaciones inestables tales como: si se tienen dos amigos pero son enemigos entre sí, o bien un grupo de tres usuarios en los cuales todos son enemigos de todos.

Analizando estas relaciones en un modelo en el que los signos de las aristas son aleatorios obtenemos el siguiente resultado: las relaciones balanceadas son más frecuentes que en el caso aleaotiro y las relacioenes desalanceadas son menos frecuentes auqneu en el caso de las tres enemistades en realidad la desviación es mínima.

El caso más dramático es el tríangulo no balanceado en donde A es amigo de B y C pero B y C son enemigos. Esto es lógico ya que va en contra del mecanismo de clausura triangular que se ha estudiado y es tan popular en las redes sociales.

El lenguaje en las redes sociales

En una red social se puede estudiar no sólo la topología de la misma sino el contenido que los usuarios escriben. Lo que se quiere estudiar es el lenguaje utilizado en estos posts. Los usuarios tienden a copiar el lenguaje de otros usuarios y esto ocurre de forma involuntaria.

Se puede hacer un promedio de todos los mensajes de la red y esto nos dará el modelo de lenguaje promedio d ela red social. Entonces se podrá comparar el lenguaje de un determinado usuario contra el lenguaje d ela red a lo largo del tiempo. Lo que se observa es que los usuarios se van adapatando al lenguaje de la red y la distancia entre el lenguaje del suaurio y el lenguaje promedio de la red va disminuyendo a lo largo del tiempo.

Esto ocurre hasta que en un determinado instante el usuario empieza a separarse del lenguaje promedio de la red. El motivo por le cual esto puede ocurrir es porque el usuairo queda atrapado en el pasado, su lenguaje evoluciona hasta un punto tal en que ya no puede modificarlo y apartir de ahísus posts va estar más lejos del lenguaje promedio de la red.

Algo muy interesante es que esto es independiente de cuanto tiempo vaya a permanecer el usuario en la red. En el momento en que el lenguaje del usuario comienza a distanciarse del promedio sus días están contados.

Sistemas de recomendación

Un sistema de recomendaciones tiene como objativo descubrir contenidos que puedan ser interesantes para el usuario.

Los sistemas de recomendaciones en general tienen el eneorme objetivo de transofrmar la forma en la que los usuarios utilizan internet cambiando un modelo basado en búsquedas por un modelo basado en descubrimiento. En un modelo basado en descubrimiento un sistema de recomendaciones descubre contenidos que pueden ser de interés para el usuairo y se los presenta: en lugar de hacer que el usuario busque los contenidos hacemos que estos lleguen al usuario.

En un modelo de descubrimiento el usuario encuentra cosas que **no sabía que existían**. Sin un sistema de recomendaciones adecuado los usuarios no podrían acceder a los contenidos que no conocen, quedando limitados sólamente a lo conocido.

Características de un sistema de recomendaciones

- **Precisión**: apunta a recomendarle al usuario contenidos que sean de su interés.
- **Serendipity**: consiste en no mostrarle al usuairo contenidos que ya conoce o que puede descubrir fácilmente por sus propios medios. Esto es esquivalente a decir que el sistema debe ser capaz de recomendarle al usuario contenidos que no son muy populares en general. Se verifica como ley que existen unos pocos productos que son muy populares y una enorme cantidad de contenidos que tiene baja popularidad. Esto se conoce como el principio de **the long tail**.
- **Diversidad**: apunta a no mostrarle al usuario contenidos que son exclusivemente de un mismo tipo.

Otros desfíos de los sistemas de recomendación

- Los gustos de los usuarios pueden cambiar. Que el sistema sea diverso puede mitigar este problema.
- La influencia del tiempo.
- El usuario a veces quiere ver cosas que no le gustan.
- Lo que el usuario califica vs lo que el usuario ve.

Sistemas basados en contenidos

En un sistema basado en contenidos el objetivo es recomendarle al usuario item similares a quellos que le han gustado.

Para cada ítem es necesario construir un it profile, un vector que represetna las características del item. Este vector puede tener muchísimas dimensiones.

Por cada usuario podemos también costruir un profile que es un vector de igual cantidad de dimensiones, luego cuando un califica una película sumamos el puntaje a quellas dimensiones para las cuales la película tiene un 1 su feature. Teniendo en cuenta cada item y de cada usairo podemos eestimar la calificación un usuario para el ítem calculando el coseno entre los perfiles:

$$cos\theta = \frac{xy}{||x||\ ||y||}$$

Podemos normalizar a todos los vectores para que tengan norma =1 y entonces el resultado dle coseno es una semejanza entre 0 y 1. Si la semejanza va de 0 a 1 podemos multiplicar la clasificación máxima para estimar una clasificación.

Ventajas de este sistema:

- Solo necesitan información del usuario para realizarle recomendaciones.
- Tiene la habilidad de detectar los gustos particulares de los usuarios.
- Pueden recomendar items que sean nuevos o no sean populares, solo se necesita el profile del item.
- Pueden explicar el motvio de sus recomendaciones listando cuales son los features que el ítem sea recomendado.

Desventajas de este sistema:

- Encontrar los features necesarios para construir los perfiles es algo realmente dificil.
- No puede recomendar nada a los usuarios nuevos ya que no tiene profile.
- No pueden recomendar items que no están dentro de los gustos de usuario. Quedan sistemas sobre especializados.
- no tienen la posibilidad de usarar la opinión de otros usuarios sobre los items.

Estos sitemas pueden funcionar muy bien pero son difíciles de crear y mantenener, principalmente por la necesidad de seleccionar todos los features necesarios para contruir un profile.

Collaborative Filtering

Sean n usuarios y m ítems. Los usuarios clasifican los ítems con un número del 1 al 5, podemos reprentar esto con una matriz de nxm que se denomina **matriz de utilidad.**

Notemos que la matriz tiene muchos huecos, esto se debe a que de todas las calificaciones posibles sólo existen algunas pocas. Esto es en cierta forma similar a decir qu ela amtriz es dispersa pero no es exacatamente lo mismo: **los elementos que faltan no son ceros sino que son números desconocidos.**

El objetivo de Collaborative Filtering es estimar las clasificaciones que nos faltan en la matriz de utillidad.

Semejanza entre los usuarios (User User)

Supongamos que queremos estimar las clasificaciones que le faltan al usaurio i. Lo que hacemos es buscar los usuairos más similares a i y luego estimar las clasifiaciones faltantes en base a un promedio ponderado de las clasificaciones de los demás usuarios poderadas de acuerdo a la semejanza que tengan con nuestro usuario i.

Necesitamos una función de semejanza que calcule cuan similares son dos ususairos en base a las clasificaciones que han realizado los mismo. Para ello lo que se hace es centrar todas las clasificaciones de los usuarios en cero, es decir, restarle a cada clasificación el promedio del usuario. Y luego, calcular la semejanza con el coseno de los vectores. Luego para calcular la clasificacion faltante lo que hacemos es:

$$r_{xi} = \frac{\sum_{y \epsilon N} s_{xy} r_{yi}}{\sum_{y \epsilon N} s_{xy}}$$

en donde N es el conjunto de los N usuarios más similares al que queremos estimar y r_{yi} es el rating del usuario y para la película i.

Nota de implementación: para tomar los N usuarios más parecidos al usuario al cuál le queremos recomendar podemos usar LSH como sabemos implica tiempo sublineal.

Semejanza entre items (item item)

Lo que vamos a hacer para estimar la clasificación de un usuario para un ítem es buscar los ítems más parecidos al que queremos estimar y que haya sido clasificado por el usuario. Luego estimamos la calificación haciendo un promedio ponderado entre las calificaciones del usuario para estos ítems y la semejanza que tienen ocn el item que queremos estimar.

El primero paso es calcular la semejanza de todas las demás películas con respecto a la uno. Para esto restamos a cada clasifiación el promedio de la pelicula y usamos el coseno.

En general el modelo item item da mejores resultados que el modelo user user. Esto puede explicarse pensando que los usuarios tienen gustos muy diversos.

Colaborative Filtering en base a desviaciones

Estimaremos la clasificación del ususario i a la película j de la siguiente forma:

$$r_{ij} = \mu + \delta_i + \delta_j + \delta ij$$

Donde:

- μ es el promedio de todas las calsificaciones de todas las películas (promedio global).
- δ_i es la desviación del usuario i con respecto a μ
- δ_j es la desviación del item j con respecto a μ
- δ_{ij} es la desviación de ususario i para la película j que se calcula como:

$$\delta_{ij} = \frac{\sum_{j_\epsilon N} s_{ij}(r_{ij} - b_{xj})}{\sum_{j \epsilon N} s_{ij}}$$

  con:

$$b_{xj} = \mu + \delta_x + \delta_j$$

La desviación del usuario a la película se calcula como que tan severo es el usuario y es la diferencia entre el promedio de clasificaciones del ususaior y el promedio global.

La desviación del item es que tan buena o mala es la película respecto al global.

Resumen:

- promedio general
- desvío del usuario al general
- desvío de la pelicula que quiero predecir al general
- promedio ponderado del desvío de las peliculas similares al general
- el promedio ponderado lo haces calculando los desvíos del usuario y los desvíos de las peliculas que ya conocemos del general
- y lo poneramos con la semejanza que obtuvimos en item item

Evaluación de sistemas de recomendación

Evaluación basada en las predicciones

Esta métrica se basa en realizar predicciones para los ítems que los usuarios ya han calificado y luego comparar estas predicciones con los ratings conocidos.

El error puede calcularse con el error cuadrático medio:

$$J = \sqrt{\frac{1}{N} \sum_{ij} (\widehat{r}_{ij} - r_{ij})^2}$$

Evaluación basada en el orden de las recomendaciones

En esta familia de métricas el sistema de recomendaciones realiza predicciones y las mismas se ordenan de mayor a menor, luego comparamos el orden de los items recomendados con el orden de los item conocidos por el usuario.

MPR: mean reciprocal rank

En esta métrica tenemos que para cada usuario una lista de items relevantes, una vez que el sistema realiza sus recomendaciones buscamos en qué el número de orden que aparece el primer ítem más releveante y calculamos 1 sobre esa posición como valor de la métrica. El promedio para todos los usuarios es la evaluación del sistema:

$$MRR(O,U) = \frac{1}{n}\sum_{u\epsilon U}\frac{1}{p_u}$$

donde p_u es la posición en la lista de recomendaciones del primer item que sabemos que es relevante para el usuario.

Mean average precision

Aquí usamos los conceptos de precisión (cantidad de ítems relevantes sobre la cantidad de items recomendados) y de recall (cantidad de items relevantes sobre el total de items relevante). Si calculamos un promedio de la precisión para cada nivel de recall diferente obtenemos la métrica conocida como average precision. El promedio de esta mérica para todos los usuarios nos da MAP.

Correlación de Spearman:

En collaborative filtering hemos usado el coeficiente de correlación de pearson para decidir la semjanza entre ususarios o items, comparando los valores d elos ratings. El coeficiente de Spearman essimilar pero en lugar de los valores de los rating socmparamos la posición o ranking de los valores. Una vez que transformamos los valores en posiciones clacculamos la correlación igual que lo hacíamos con la fórmula de pearson.

El principal problema de la fórmula de Spearman es que castiga a todos los errores de la misma forma. En general queremos que un error en el ordenamiento de los item recomendados sea más grave cuanto más chico es el ranking del ítem.

Normalized Discounted Cumulative Gain

En primer lugar vemos como calcular DCG (discounted cumulative gain), la idea es muy simple lo que hacemos es sumar la división del rating real (puntaje) de cada ítem por el logaritmo de su posición en la lista de recomendaciones, pero para los dos primero ítems no dividimo o dividimos por 1.

Para normalizar y obtener la métrica nDCG lo que vamos a hacer es calcular cuál sería el valor de DCG si nuestro sistema de recomendación hubiese devuelto los ítem ordenados de acuerdo a las calificaciones de usuario.

Simplemente calculamos DCG/perfect DCG para obtener nDCG. Luego, simplemente promediamos el nDCG par atodos lo usuarios para obtener un promedio para el sistema de recomendaciones.

Modelos Latentes

Modelo latente: porque nos permite inferir preferencis de los usuarios que no están explícitas en la matriz de utilidad.

SVD ++

Está basado en el simple concepto de factorizar l amatriz de utilidad para descubir los modelos latentes. El principio aquí es : descomponer la matriz de utldad en producto de dos matrices Q y P. En donde Q tendrá tantas filas como items y tantas columnas como factores latentes usemos y P tendrá tantas filas como usuarios y tantas columnas como factores.

$$U_{mxn} = Q_{mxr} P^t_{nxr}$$

La cantidad de factores r es un parámetro que debemos determinar.

Es decir que en Q vamos a poder observar que tipo de factores están asociados a cada película y en P tendremos las preferencias de los usuarios con respecto a los factores.

El objetivo es encontrar Q y P tales que Q*R minimice el error con la matriz de utilidad original.

Si logramos hallar estas matrices Q y p entonces podemos estimar la calificación de ususario i para la película j haciendo $q_i * p_j$

Lo interesante de esto es que las matrices Q y P que minimicen el error con la original nos dan mágicamente todos los valores que nos faltaban en la matriz de utilidad original.

Este es un problema muy similar al que tenemos con LSI sin embargo no podemos usar la SVD pq no tenemos completa la matriz original.

Planteamos el siguiente problema de optimización:

$$min_{P,Q} \sum_{(i,x)\epsilon R} (R_{xi} - q_i p_x)^2$$

r que usamos es nuestro set de entrenamiento. Podríamos enontrar valores para las matrices Q y P que minimicen la diferencia con la matriz de utilidad de nuestro set de entrenamiento pero que funcionen muy mal para el set de pruebas. Por lo tanto tenemos Overfitting. Podemos resolver este problema mediante una regularización. Implica modificar el problema de optimización de antes y buscar los parámetros por el método Gradient Descent. No quedaría dos nuevos hiper parámetros.

Factorization machine

Este algoritmo puede usarse tanto para recomendaciones como para clasificación o regresión y se basa en encontrar los factores latentes asociados a cada variable del sistema.

Supongamos que queremos hacer una regresión lineal con user id, movie id, y rating:

Usamos one hot ecoding para user id y movie id. Esto nos lleva a un modelo de regresión lineal de la forma:

$$\widehat{y} = w_0 + \sum_{i=1}^{n} w_i * x_i + \sum_{j=1+n}^{m+n} w_j * xj$$

siendo m la cantidad de películas y n la cantidad de usuarios.

Este modelo captura un coeficiente por cada una de las variables del modelo y estima la calificación en base a estos coeficientes. En otras palabras la calificación va a depender del usuario y la película.

Podemos extender este modelo agregando otras variables codificadas mediante one-hot-ecoding como por ejemplo: cuáles son otras películas calificadas por el usuario, en qué momento hizo la calificación o cuál es la última película calificada.

Estos vectores tienen cientos de miles o millones de dimensiones. Con este modelo podríamos crear una regresión lineal usando todos los features, este modelo sin embargo no capturaría la interacción entre las diferentes variables. No tenemos un coeficiente que relacione usuario y película.

Al factorizar lo que hacemos es agregar todos los coeficienttes de interacción entre cada una de als variables de modo en que en total agreguemos n2 nuevas variables del tipo w_ij indicando la relación entre la variable i y la variable j del modelo.

$$\widehat{y} = w_0 + \sum_{i=1}^{n} w_i * x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} * x_i * x_j$$

Este modelo es muy completo pero tiene el problema que la cantidad de coeficientes sugre de un problema de explosión combinatoria. La solución a esto es la siguiente: la idea es representar cada variable (columna) de nuestro modelo mediante un vector de k elementos siendo k la cantidad de factores latentes que queremos usar. El modelo se reescribe de la siguiente manera:

$$\widehat{y} = w_0 + \sum_{i=1}^{n} w_i * x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < v_i, v_j > * x_i * x_j$$