AMAT- Ciencia de Datos y Machine Learning 2

Karina Lizette Gamboa Puente Oscar Arturo Bringas López

Contents

Chapter 1

BIENVENIDA

1.1 Objetivo

Desarrollar conocimiento y habilidades para implementar modelos complejos de Machine Learning a través de un flujo de trabajo limpio, ordenado y sistematizado a mediante las librerías en *R* más novedosas que han sido desarrolladas hasta el momento. Al finalizar este curso, el participante será capáz de combinar distintas clases de modelos para dar una solución compleja y precisa a problemas predictivos. Aprenderá a cuantificar los problemas éticos asociados al sesgo o inequidad producidos por modelos de machine learning, así como su interpretación en el mundo productivo. Finalmente, se estudiará la manera de desarrollar un diseño de experimento para implementarse en el ámbito empresarial de modo que el participante pueda tomar mejores decisiones para contribuir en su ambiente laboral.

Se asume que el alumno tiene conocimientos generales de estadística, bases matemáticas y de programación básica en R y que cuenta con los conocimientos teóricos básicos de machine learning y prácticos con tidymodels.

1.2 Instructores

ACT. ARTURO BRINGAS

LinkedIn: arturo-bringas **Email:** act.arturo.b@ciencias.unam.mx

Actuario, egresado de la Facultad de Ciencias y Maestría en Ciencia de Datos, ITAM. Experiencia en modelos predictivos y de clasificación de machine learning aplicado a seguros, deportes y movilidad internacional. Es jefe de departamento

en Investigación Aplicada y Opinión de la UNAM, donde realiza estudios estadísticos de impacto social. Es consultor para empresas y organizaciones como GNP, El Universal, UNAM, Sinnia, la Organización de las Naciones Unidas Contra la Droga y el Delito (UNODC), entre otros. Actualmente es profesor de *ciencia de datos y machine learning* en AMAT y se desempeña como consultor independiente en diferentes proyectos contribuyendo a empresas en temas de machine learning, estadística, series de tiempo, visualización de datos y análisis geoespacial.

ACT. KARINA LIZETTE GAMBOA

LinkedIn: KaLizzyGam **Email:** lizzygamboa@ciencias.unam.mx

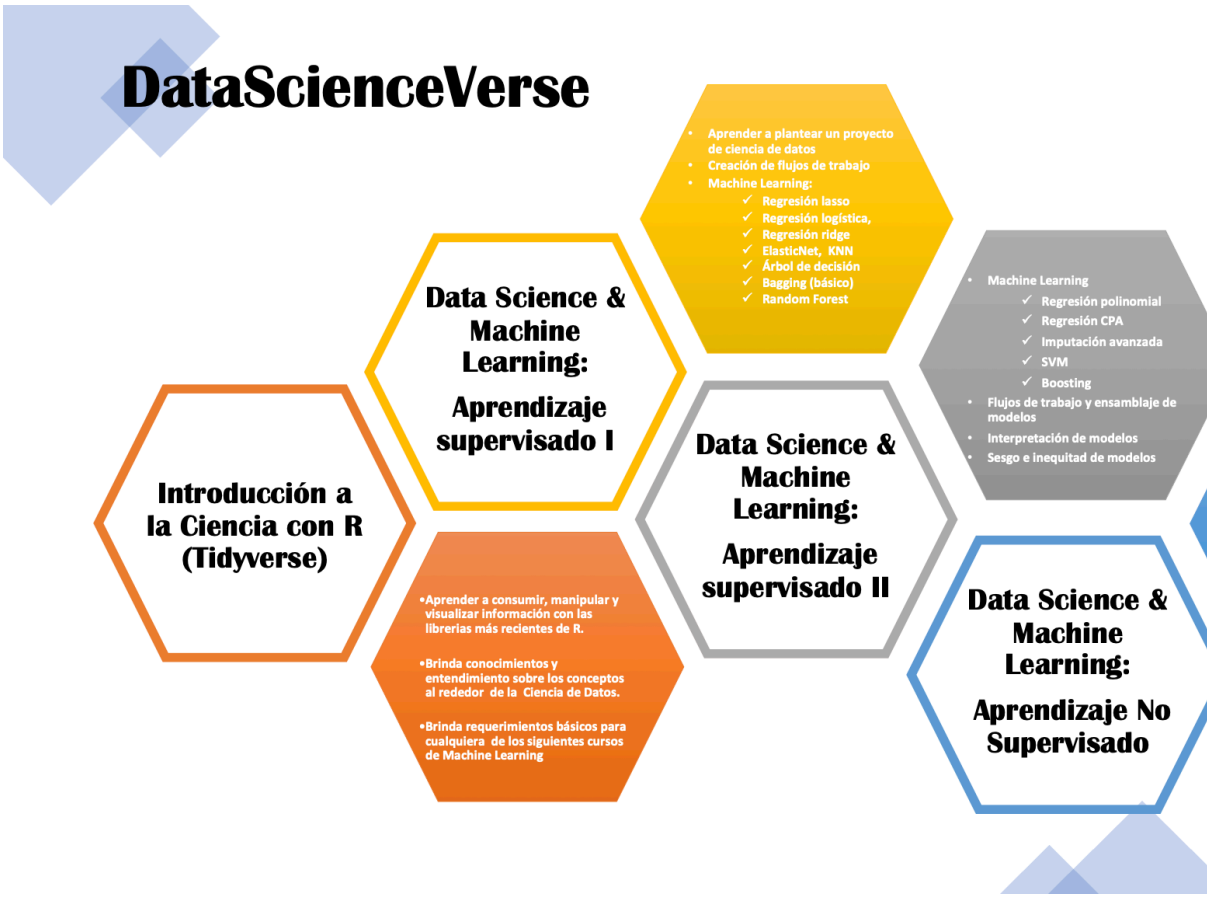
Actuaria, egresada de la Facultad de Ciencias, UNAM, candidata a Maestra en Ciencia de Datos por el ITAM.

Experiencia en áreas de analítica predictiva e inteligencia del negocio. Lead y Senior Data Scientist en consultoría en diferentes sectores como tecnología,

asegurador, financiero y bancario. Experta en entendimiento de negocio para la correcta implementación de algoritmos de inteligencia y explotación de datos. Actualmente se desarrolla como Arquitecta de Soluciones Analíticas en Merama, startup mexicana clasificada como uno de los nuevos unicornios de Latinoamérica. Senior Data Science en CLOSTER y como profesora del diplomado de Metodología de la Investigación Social por la UNAM así como instructora de cursos de Ciencia de Datos en AMAT.

Empresas anteriores: GNP, Activer Banco y Casa de Bolsa, PlayCity Casinos, RakenDataGroup Consulting, entre otros.

1.3 Ciencia de Datos en R



1.4 Estructura del curso actual

1.4.1 Alcances del curso

Al finalizar el módulo, el participante sabrá plantear un proyecto de ciencia de datos, desde sus requerimientos hasta sus implementación comercial. Sabrá crear flujos de trabajo limpios y ordenados para crear poderosos modelos de Machine Learning. Podrá comparar múltiples modelos y seleccionar el que más aportación realice a su negocio considerando la ética alrededor del sesgo e inequidad producida por modelos. Profundizará su conocimiento en la interpretación de modelos complejos y aprenderá a cuantificar el beneficio comercial de la implementación de modelos.

Requisitos:

Computadora con al menos 4Gb Ram.

Instalación de R con al menos versión 4.1.0

Instalación de Rstudio con al menos versión 1.4

Data Science & Machine Learning (Aprendizaje Supervisado I)

Temario:

1.- Machine Learning (10 HRS)

- Regresión polinomial
- Regresión con CPA
- Imputación
- SVM
- Boosting

2. Flujos de trabajo y ensamblajes (8 HRS)

- Pipelines
- Workflowsets
- Comparación de modelos
- Stacking

3. Sesgo e inequidad de modelos (4 HRS)

- Cuantificación de sesgo
- Cuantificación de inequidad

4. Interpretación de modelos (4 HRS)

- LIME
- ghv

5. Aplicación a negocios (6 HRS)

- Diseño de experimentos en campañas de retención
- Valuación de implementación de modelos

1.5 Duración y evaluación del curso

- El programa tiene una duración de 32 hrs.
- Las clases serán impartidas los días domingo, de 9:00 am a 1:00 pm
- Serán asignados ejercicios que el participante deberá resolver entre una semana y otra.
- Al final del curso se solicitará un proyecto final, el cual **deberá ser entregado para ser acreedor a la constancia de participación**.

1.6 Recursos y dinámica de clase

En esta clase estaremos usando:

- R da click aquí si aún no lo descargas
- RStudio da click aquí también
- Miro úsame
- Zoom Clases
  - Pulgar arriba: Voy bien, estoy entendiendo!
  - Pulgar abajo: Eso no quedó muy claro
  - Mano arriba: Quiero participar/preguntar ó Ya estoy listo para iniciar
- Grupo de WhatsApp El chismecito está aquí
- Google Drive
- Notas de clase Revisame si quieres aprender

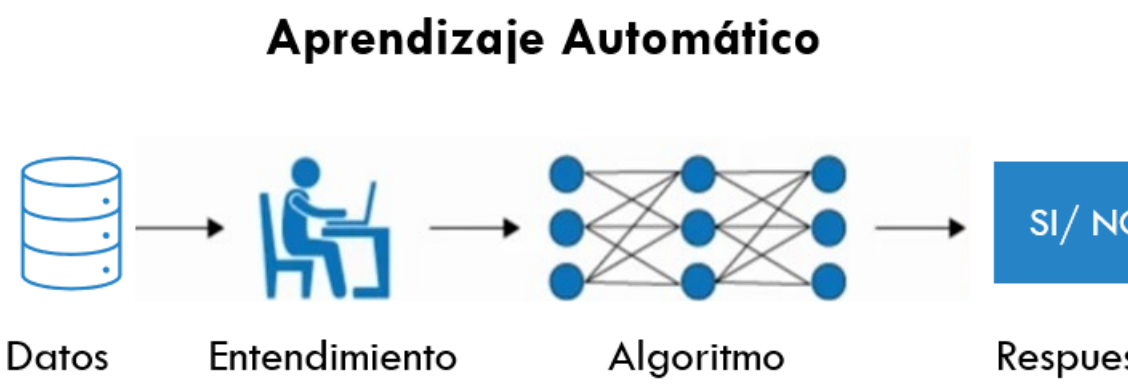
Chapter 2

Repaso

En los cursos anteriores, hemos hablando a cerca del proceso completo de Ciencia de Datos, para poder empezar con la segunda parte del curso de Analisis Supervisado, valele la pena hacer un breve repaso de lo que hemos estudiado hasta el momento.

2.1 Machine Learning

Machine Learning o –aprendizaje automático– es una rama de la inteligencia artificial que permite que las máquinas aprendan de los patrones existentes en los datos. Se usan métodos computacionales para aprender de datos con el fin de producir reglas para mejorar el desempeño en alguna tarea o toma de decisión. (Está enfocado en la programación de máquinas para aprender de los patrones existentes en datos principalmente estructurados y anticiparse al futuro)


Aprendizaje Automático
Datos
Entendimiento
Algoritmo
SI/ NO

2.2 Tipos de aprendizaje

Platicamos en el módulo pasado que al hablar de Machine Learning, existen distintos tipos de aprendizaje, siendo los más comúnes:

- Aprendizaje supervisado
- Aprendizaje no supervisado

Otreos ejemplos de especialidades son

- Aprendizaje profundo
- Aprendizaje por refuerzo

La diferencia entre el análisis supervisado y el no supervisado es la etiqueta, es decir, en el análisis supervisado tenemos una etiqueta "correcta" y el objetivo de los algoritmos es predecir esta etiqueta.

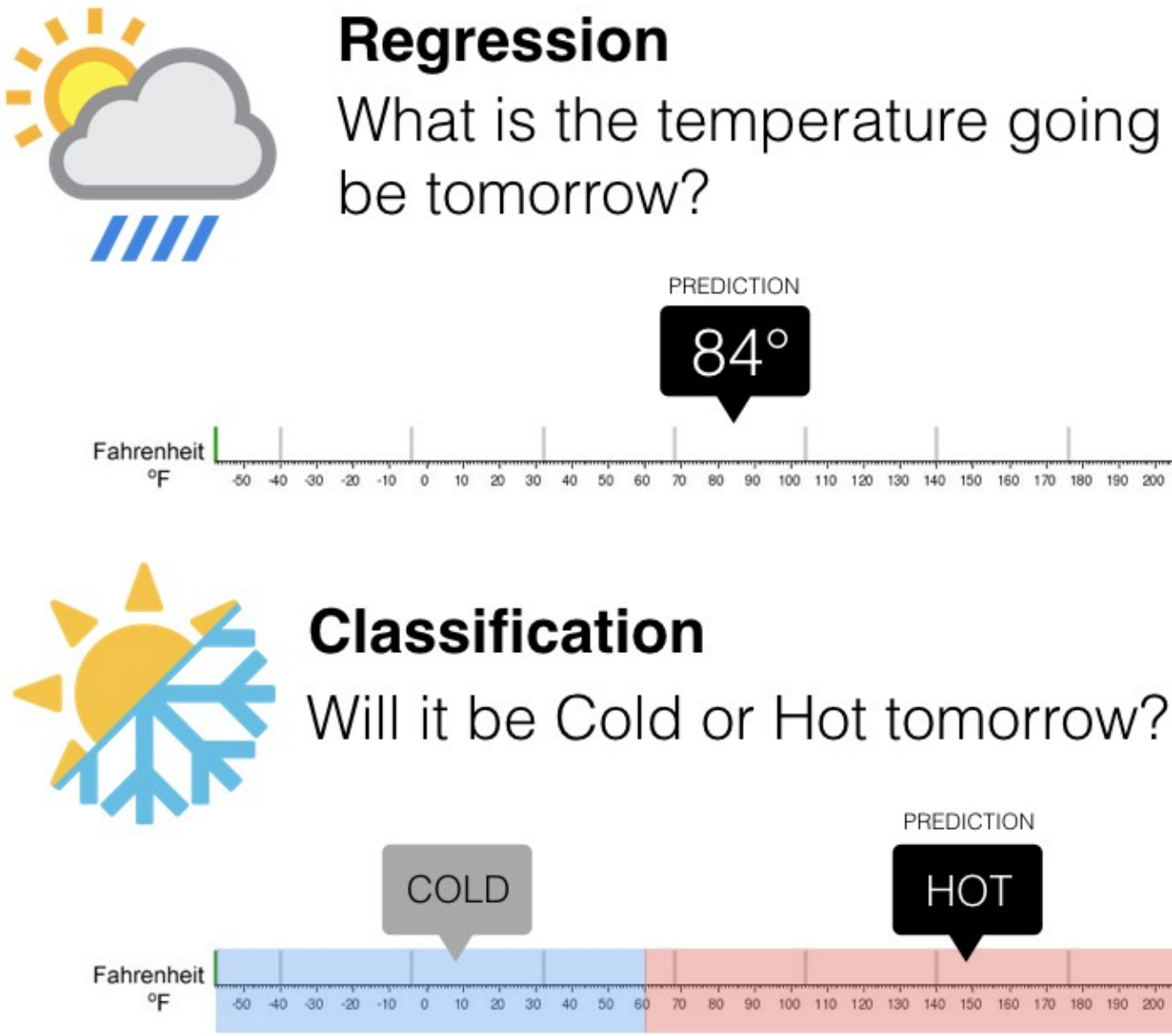
2.2.1 Aprendizaje supervisado

- Conocemos la respuesta correcta de antemano.
- Esta respuesta correcta fue "etiquetada" por un humano (la mayoría de las veces, en algunas circunstancias puede ser generada por otro algoritmo).
- Debido a que conocemos la respuesta correcta, existen muchas métricas de desempeño del modelo para verificar que nuestro algoritmo está haciendo las cosas "bien".

2.2.1.1 Tipos de aprendizaje supervisado (Regresión vs clasificación)

Existen dos tipos principales de aprendizaje supervisado, esto depende del tipo de la variable respuesta:

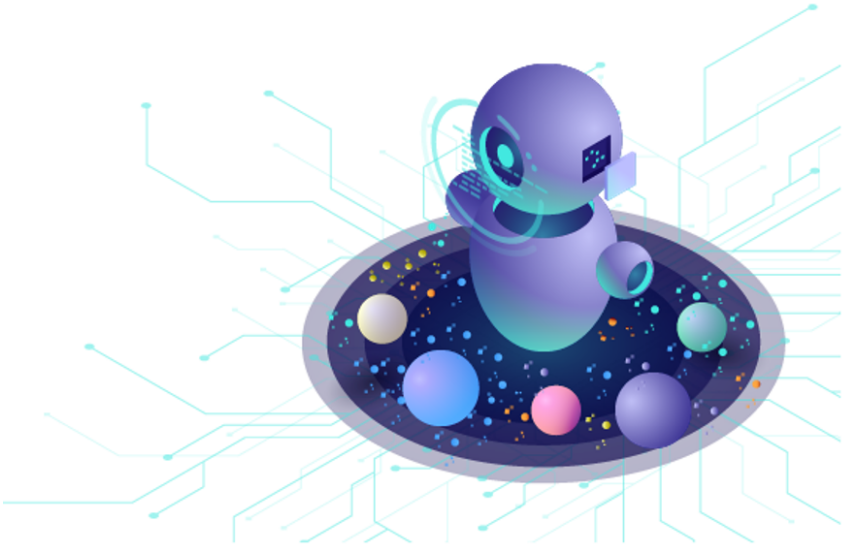
- Los algoritmos de **clasificación** se usan cuando el resultado deseado es una etiqueta discreta, es decir, clasifican un elemento dentro de diversas clases.
- En un problema de **regresión**, la variable target o variable a predecir es un valor numérico.



2.2.2 Aprendizaje no supervisado

- Aquí no tenemos la respuesta correcta de antemano ¿cómo podemos saber que el algoritmo está bien o mal?
- Estadísticamente podemos verificar que el algoritmo está bien

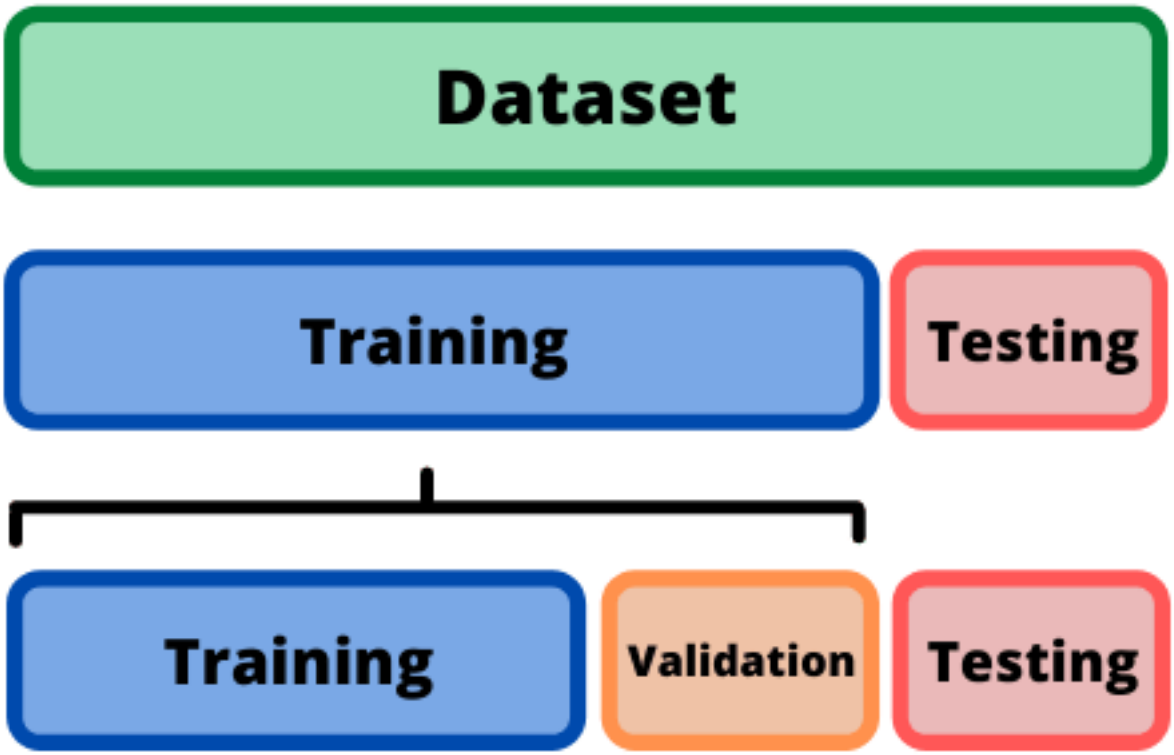
- Siempre tenemos que verificar con el cliente si los resultados que estamos obteniendo tienen sentido de negocio. Por ejemplo, número de grupos y características



2.3 El proceso de Machine Learning

La cuestión no es solo saber para qué sirve el Machine Learning, sino que saber cómo funciona y cómo poder implementarlo en la industria para aprovecharse de sus beneficios. Hay ciertos pasos que usualmente se siguen para crear un modelo de Machine Learning. Estos son típicamente realizados por científicos de los datos que trabajan en estrecha colaboración con los profesionales de los negocios para los que se está desarrollando el modelo.

1. **Seleccionar y preparar un conjunto de datos de entrenamiento**: Los **datos de entrenamiento** son un conjunto de datos representativos de los datos que el modelo de Machine Learning ingerirá para resolver el problema que está diseñado para resolver. Los datos de entrenamiento deben prepararse adecuadamente: aleatorizados y comprobados en busca de desequilibrios o sesgos que puedan afectar al entrenamiento. También deben dividirse en dos subconjuntos: el **subconjunto de entrenamiento**, que se utilizará para entrenar el algoritmo, y el **subconjunto de validación**, que se utilizará para probarlo y perfeccionarlo.



2. **Elegir un algoritmo para ejecutarlo en el conjunto de datos de entrenamiento**: Este es uno de los pasos más importantes, ya que se debe elegir qué algoritmo utilizar, siendo este un conjunto de pasos de procesamiento estadístico. El tipo de algoritmo depende del tipo (supervisado o no supervisado), la cantidad de datos del conjunto de datos de entrenamiento y del tipo de problema que se debe resolver.

3. **Entrenamiento del algoritmo para crear el modelo**: El entrenamiento del algoritmo es un proceso iterativo: implica ejecutar las variables a través del algoritmo, comparar el resultado con los resultados que debería haber producido, ajustar los pesos y los sesgos dentro del algoritmo que podrían dar un resultado más exacto, y ejecutar las variables de nuevo hasta que el algoritmo devuelva el resultado correcto

la mayoría de las veces. El algoritmo resultante, entrenado y preciso, es el modelo de Machine Learning.

4. **Usar y mejorar el modelo**: El paso final es utilizar el modelo con nuevos datos y, en el mejor de los casos, para que mejore en precisión y eficacia con el tiempo. De dónde procedan los nuevos datos dependerá del problema que se resuelva. Por ejemplo, un modelo de Machine Learning diseñado para identificar el spam ingerirá mensajes de correo electrónico, mientras que un modelo de Machine Learning que maneja una aspiradora robot ingerirá datos que resulten de la interacción en el mundo real con muebles movidos o nuevos objetos en la habitación.

2.4 Errores: Sesgo vs varianza

En el mundo de Machine Learning cuando desarrollamos un modelo nos esforzamos para hacer que sea lo más preciso, ajustando los parámetros, pero la realidad es que no se puede construir un modelo 100% preciso ya que nunca pueden estar libres de errores.

Comprender cómo las diferentes fuentes de error generan sesgo y varianza nos ayudará a mejorar el proceso de ajuste de datos, lo que resulta en modelos más precisos, adicionalmente también evitará el error de sobreajuste y falta de ajuste.

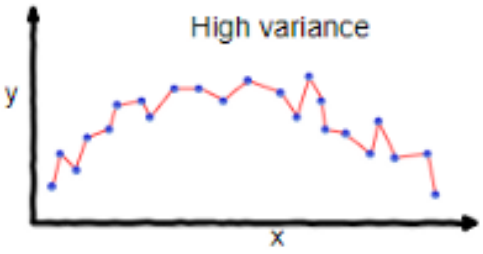
- **Error por sesgo:**

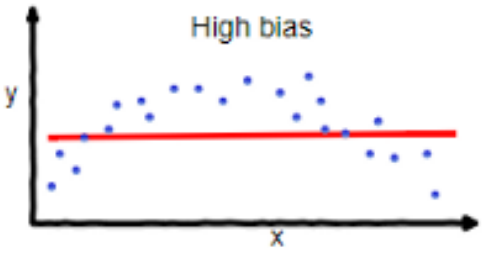
Es la diferencia entre la predicción esperada de nuestro modelo y los valores verdaderos. Aunque al final nuestro objetivo es siempre construir modelos que puedan predecir datos muy cercanos a los valores verdaderos, no siempre es tan fácil porque algunos algoritmos son simplemente demasiado rígidos para aprender señales complejas del conjunto de datos.

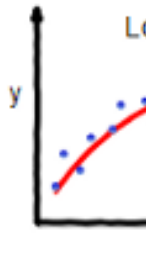
Imagina ajustar una regresión lineal a un conjunto de datos que tiene un patrón no lineal, no importa cuántas observaciones más recopiles, una regresión lineal no podrá modelar las curvas en esos datos. Esto se conoce como *underfitting*.

- **Error por varianza:**

Se refiere a la cantidad que la estimación de la función objetivo cambiará si se utiliza diferentes datos de entrenamiento. La función objetivo se estima a partir de los datos de entrenamiento mediante un algoritmo de Machine Learning, por lo que deberíamos esperar que el algoritmo tenga alguna variación. Idealmente no debería cambiar demasiado de un conjunto de datos de entrenamiento a otro.







overfitting underfitting G

Los algoritmos de Machine Learning que tienen una gran varianza están fuertemente influenciados por los detalles de los datos de entrenamiento, esto significa que los detalles de la capacitación influyen en el número y los tipos de parámetros utilizados para caracterizar la función de mapeo.

- **Error irreducible:** El error irreducible no se puede reducir, independientemente de qué algoritmo se usa. También se le conoce como ruido y, por lo general, proviene por factores como variables desconocidas que influyen en el mapeo de las variables de entrada a la variable de salida, un conjunto de características incompleto o un problema mal enmarcado. Acá es importante comprender que no importa cuán bueno hagamos nuestro modelo, nuestros datos tendrán cierta cantidad de ruido o un error irreductible que no se puede eliminar.

2.5 Ingeniería de características (Feature Engineering)

La ingenería de datos y procesamiento de datos es parte vital del desarrollo de un buen modelo.

En este curso analizaremos distintos métodos de machine learning que permitirán predecir una respuesta numérica o categórica. Usaremos el lenguaje de programación *R* para dicho procesamiento.

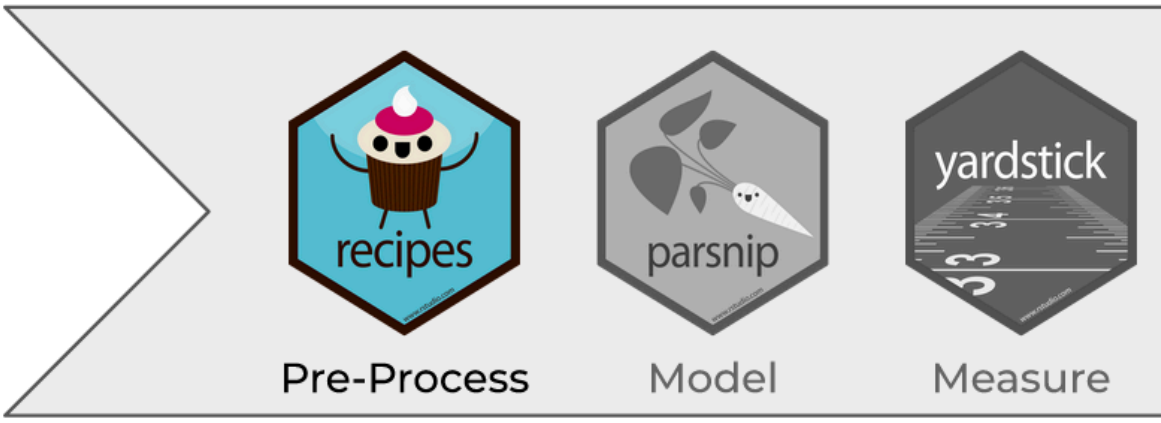
Hay varios pasos que se deben de seguir para crear un modelo útil:

- Recopilación de datos.
- Limpieza de datos.
- Creación de nuevas variables.
- Estimación de parámetros.
- Selección y ajuste del modelo.
- Evaluación del rendimiento.

Al comienzo de un proyecto, generalmente hay un conjunto finito de datos disponibles para todas estas tareas.

OJO: A medida que los datos se reutilizan para múltiples tareas, aumentan los riesgos de agregar sesgos o grandes efectos de errores metodológicos.

2.5.1 Pre-procesamiento de datos



Como punto de partida para nuestro flujo de trabajo de aprendizaje automático, necesitaremos datos de entrada. En la mayoría de los casos, estos datos se

cargarán y almacenarán en forma de *data frames* o *tibbles* en R. Incluirán una o varias variables predictoras y, en caso de aprendizaje supervisado, también incluirán un resultado conocido.

Sin embargo, no todos los modelos pueden lidiar con diferentes problemas de datos y, a menudo, necesitamos transformar los datos para obtener el mejor rendimiento posible del modelo. Este proceso se denomina pre-procesamiento y puede incluir una amplia gama de pasos, como:

- **Dicotomización de variables:** Variables cualitativas que solo pueden tomar el valor 0 o 1 para indicar la ausencia o presencia de una condición específica. Estas variables se utilizan para clasificar los datos en categorías mutuamente excluyentes o para activar comandos de encendido / apagado

- **Near Zero Value (nzv) o Varianza Cero:** En algunas situaciones, el mecanismo de generación de datos puede crear predictores que solo tienen un valor único (es decir, un "predictor de varianza cercando a cero"). Para muchos modelos (excluidos los modelos basados en árboles), esto puede hacer que el modelo se bloquee o que el ajuste sea inestable.

De manera similar, los predictores pueden tener solo una pequeña cantidad de valores únicos que ocurren con frecuencias muy bajas.

- **Imputaciones:** Si faltan algunos predictores, ¿deberían estimarse mediante imputación?

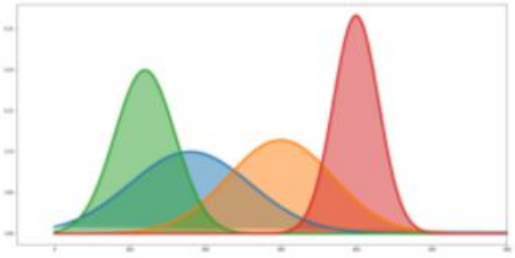
- **Des-correlacionar:** Si hay predictores correlacionados, ¿debería mitigarse esta correlación? Esto podría significar filtrar predictores, usar análisis de componentes principales o una técnica basada en modelos (por ejemplo, regularización).

ID	MUJER	HOMBRE
1	1	0
2	0	1
3	0	1
4	0	1
5	0	1
6	1	0

ID	MUJER	HOMBRE
1	1	0
2	0	1
3	0	
4	0	
5	0	
6	1	0

- **Normalizar:** ¿Deben centrarse y escalar los predictores?

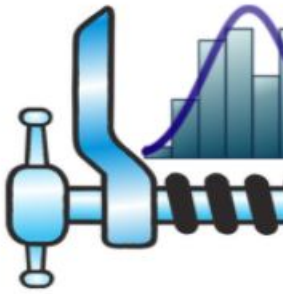
¿Por qué estandarizar?



La estandarización es la forma en la que pueden compararse diferentes métricas y escalas para hallar relaciones con sentido y uniformidad. La pregunta que resuelve es ¿que tan por encima y qué tan abajo está un valor de la media?

Por ejemplo: Calificaciones de diferentes materias con varias escalas diferentes.

¿Por qué re-escalar?



El re-escalamiento entre 0 y 1 fu
poner el valor máximo en el 1 y
manera que entre 0 y 1 puede m
de la medición

Por ejemplo: En lugar de te
calificaciones entre 5 y 10. La cali
al valor del 1 dentro del escalamie

- **Transformar:** ¿Es útil transformar los predictores para que sean más simétricos? (por ejemplo, escala logarítmica).

Dependiendo del caso de uso, algunos pasos de pre-procesamiento pueden ser indispensables para pasos posteriores, mientras que otros solo son opcionales. Sin embargo, dependiendo de los pasos de pre-procesamiento elegidos, el rendimiento del modelo puede cambiar significativamente en pasos posteriores. Por lo tanto, es muy común probar varias configuraciones.

En la tabla, ✓ indica que el método es obligatorio para el modelo y × indica que no lo es. El símbolo ◦ significa que la técnica puede ayudar al modelo, pero no es obligatorio.

model	dummy	zv	impute	decorrelate	normalize	transform
bag_mars()	✔	×	✔	○	×	○
bag_tree()	×	×	×	○[1]	×	×
boost_tree()	×+	○	✔+	○[1]	×	×
C5_rules()	×	×	×	×	×	×
cubist_rules()	×	×	×	×	×	×
decision_tree()	×	×	×	○[1]	×	×
discrim_flexible()	✔	✔	✔	✔	×	○
discrim_linear()	✔	✔	✔	✔	×	○
discrim_regularized()	✔	✔	✔	✔	×	○
linear_reg()	✔	✔	✔	✔	×	○
logistic_reg()	✔	✔	✔	✔	×	○
mars()	✔	×	✔	○	×	○
mlp()	✔	✔	✔	✔	✔	✔
multinom_reg()	✔	✔	✔	✔	×	○
naive_Bayes()	×	✔	✔	○[1]	×	×
nearest_neighbor()	✔	✔	✔	○	✔	✔
pls()	✔	✔	✔	×	✔	✔
poisson_reg()	✔	✔	✔	✔	×	○
rand_forest()	×	○	✔+	○[1]	×	×
rule_fit()	✔	×	✔	○[1]	×	×
svm_*()	✔	✔	✔	✔	✔	✔

x	x_prop
691	0.1836789
639	0.1698565
969	0.2575758
955	0.2538543
508	0.1350346

Notación:

1. Es posible que la des-correlación de predictores no ayude a mejorar el rendimiento. Sin embargo, menos predictores correlacionados pueden mejorar la estimación de las puntuaciones de importancia de la varianza. Esencialmente, la selección de predictores altamente correlacionados es casi aleatoria.

La notación + significa que la respuesta depende de la implementación:

- Teoricamente, cualquier modelo basado en árboles no requiere imputación de datos, sin embargo, muchas implementaciones de conjuntos de árboles requieren imputación.
- Si bien los métodos de refuerzo basados en árboles generalmente no requieren la creación de variables ficticias, los modelos que usan `xgboost` sí lo hacen.

2.5.2 Ingeniería de datos

La ingeniería de datos abarca actividades que reformatean los valores de los predictores para que se puedan utilizar de manera eficaz para nuestro modelo. Esto incluye transformaciones y codificaciones de los datos para representar mejor sus características importantes.

Por ejemplo:

1.- Supongamos que un conjunto de datos tiene dos predictores que se pueden representar de manera más eficaz en nuestro modelo como una proporción, así, tendríamos un nuevo predictor a partir de la proporción de los dos predictores originales.

2.- Al elegir cómo codificar nuestros datos en el modelado, podríamos elegir una opción que creemos que está más asociada con el resultado. El formato original de los datos, por ejemplo numérico (edad) versus categórico (grupo).

Edad	Grupo
7	Niños
78	Adultos mayores
17	Adolescentes
25	Adultos
90	Adultos mayores

La ingeniería y el pre-procesamiento de datos también pueden implicar el reformateo requerido por el modelo. Algunos modelos utilizan métricas de distancia geométrica y, en consecuencia, los predictores numéricos deben centrarse y escalar para que estén todos en las mismas unidades. De lo contrario, los valores de distancia estarían sesgados por la escala de cada columna.

2.6 Recetas

Una receta es una **serie de pasos o instrucciones para el procesamiento de datos.** A diferencia del método de fórmula dentro de una función de modelado, **la receta define los pasos sin ejecutarlos** inmediatamente; es sólo una especificación de lo que se debe hacer. La estructura de una receta sigue los siguientes pasos:

1. Inicialización
2. Transformación
3. Preparación
4. Aplicación

La siguiente sección explica la estructura y flujo de transformaciones:

```
receta <- recipe(response ~ X1 + X2 + X3 + ... + Xn, data = dataset ) %>%
  transformation_1(...) %>%
  transformation_2(...) %>%
  transformation_3(...) %>%
  ...
  final_transformation(...) %>%
  prep()

bake(receta, new_data = new_dataset)
```

A continuación se muestran distintos ejemplos de transformaciones realizadas comunmente en el pre-procesamiento de modelos predictivos. Como ejemplo, utilizaremos el subconjunto de predictores disponibles en los datos de vivienda: `Ames`

- Vecindario (29 vecindarios)
- Superficie habitable bruta sobre el nivel del suelo
- Año de constricción
- Tipo de edificio

ANTERIORMENTE... Un modelo de regresión lineal ordinario se ajustaba a los datos con la función estándar `lm()` de la siguiente manera:

```
lm(Sale_Price ~ Neighborhood + log10(Gr_Liv_Area) + Year_Built + Bldg_Type, data = ames
```

Cuando se ejecuta esta función, los datos se convierten en a una matriz de diseño numérico (también llamada matriz de modelo) y luego se utiliza el método de mínimos cuadrados para estimar los parámetros. Lo que hace **la fórmula anterior se puede descomponer en una serie de pasos:**

> **1.-** El precio de venta se define como el resultado, mientras que las variables de vecindario, superficie habitable bruta, año de construcción y tipo de edificio se definen como predictores.
>
> **2.-** Se aplica una transformación logarítmica al predictor de superficie habitable bruta.
>
> **3.-** Las columnas de vecindad y tipo de edificio se convierten de un formato no numérico a un formato numérico (dado que los mínimos cuadrados requieren predictores numéricos).

La siguiente receta es equivalente a la fórmula anterior:

```
simple_ames <- recipe(
  Sale_Price ~ Neighborhood + Gr_Liv_Area + Year_Built + Bldg_Type,
  data = ames) %>%
  step_log(Gr_Liv_Area, base = 10) %>%
  step_dummy(all_nominal_predictors())

simple_ames
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          4
##
## Operations:
##
## Log transformation on Gr_Liv_Area
## Dummy variables from all_nominal_predictors()
```

Ventajas de usar una receta:

- **Los cálculos se pueden reciclar entre modelos** ya que no están estrechamente acoplados a la función de modelado.
- Una receta permite un **conjunto más amplio de opciones de procesamiento** de datos que las que pueden ofrecer las fórmulas.
- La **sintaxis puede ser muy compacta**. Por ejemplo, `all_nominal_predictors()` se puede usar para capturar muchas variables para tipos específicos de procesamiento, mientras que una fórmula requeriría que cada una se enumere explícitamente.
- Todo el procesamiento de datos se puede capturar en un solo objeto en lugar de tener scripts que se repiten o incluso se distribuyen en diferentes archivos.

2.6.1 Pasos y estructura de recetas

Como se mostró anteriormente, existen 4 pasos fundamentales para el procesamiento y transformación de conjuntos de datos. Estos pasos se describen de la siguiente manera:

- **Receta:** Inicializa una receta y define los roles de las variables

- **Transformaciones:** Mutaciones a los renglones y columnas hasta desear el resultado
- **Preparación:** Se realizan las estimaciones estadísticas con los datos

La función `prep()` estima las cantidades requeridas y las estadísticas necesarias para cualquier paso declarado en la receta.

```
prep <- prep(simple_ames)

prep
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          4
##
## Training data contained 2930 data points and no missing data.
##
## Operations:
##
## Log transformation on Gr_Liv_Area [trained]
## Dummy variables from Neighborhood, Bldg_Type [trained]
```

- **Aplicación** Se llevan a cabo las transformaciones especificadas en la receta preparada a un conjunto de datos.

Finalmente, la función `bake()` lleva a cabo la transformación de un conjunto de datos a través de las estimaciones indicadas en una receta y **aplica las operaciones a un conjunto de datos para crear una matriz de diseño**. La función `bake(object, new_data = NULL)` devolverá los datos con los que se entrenó la receta.

Nota: La función `juice()` devolverá los resultados de una receta en la que se hayan aplicado todos los pasos a los datos. Similar a la función `bake()` con el comando `new_data = NULL`.

```
simple_ames %>%
  prep() %>%
  bake(new_data = NULL) %>%
  glimpse()
```

```
## Rows: 2,930
## Columns: 35
## $ Gr_Liv_Area                                          <dbl> 3.219060, 2.95230~
## $ Year_Built                                           <int> 1960, 1961, 1958,~
## $ Sale_Price                                           <int> 215000, 105000, 1~
## $ Neighborhood_College_Creek                           <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Old_Town                                <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Edwards                                 <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Somerset                                <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Northridge_Heights                      <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Gilbert                                 <dbl> 0, 0, 0, 0, 1, 1,~
## $ Neighborhood_Sawyer                                  <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Northwest_Ames                          <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Sawyer_West                             <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Mitchell                                <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Brookside                               <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Crawford                                <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Iowa_DOT_and_Rail_Road                  <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Timberland                              <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Northridge                              <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Stone_Brook                             <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_South_and_West_of_Iowa_State_University <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Clear_Creek                             <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Meadow_Village                          <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Briardale                               <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Bloomington_Heights                     <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Veenker                                 <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Northpark_Villa                         <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Blueste                                 <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Greens                                  <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Green_Hills                             <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Landmark                                <dbl> 0, 0, 0, 0, 0, 0,~
## $ Neighborhood_Hayden_Lake                             <dbl> 0, 0, 0, 0, 0, 0,~
## $ Bldg_Type_TwoFmCon                                   <dbl> 0, 0, 0, 0, 0, 0,~
## $ Bldg_Type_Duplex                                     <dbl> 0, 0, 0, 0, 0, 0,~
## $ Bldg_Type_Twnhs                                      <dbl> 0, 0, 0, 0, 0, 0,~
## $ Bldg_Type_TwnhsE                                     <dbl> 0, 0, 0, 0, 0, 0,~
```

En cuanto a las transformaciones posibles, existe una gran cantidad de funciones que soportan este proceso. En esta sección se muestran algunas de las transformación más comunes, entre ellas:

- Normalización
- Dicotomización
- Creación de nuevas columnas

- Datos faltantes
- Imputaciones
- Interacciones
- Etc.

2.6.1.1 Normalizar columnas numéricas

Quizá la transformación numérica más usada en todos los modelos es la estandarización o normalización de variables numéricas. **Este proceso se realiza para homologar la escala de las variables numéricas**, de modo que no predomine una sobre otra debido a la diferencia de magnitudes o escalas. Este proceso se tiene de fondo el siguiente proceso estadístico:

$$Z = \frac{X - \hat{\mu}_x}{\hat{\sigma}_x}$$

Donde:

- X = Es una variable o columna numérica
- $\hat{\mu}_x$ = Es la estimación de la media de la variable X
- $\hat{\sigma}_x$ = Es la estimación de la desviación estándar de la variable X

La librería recipes nos permite realizar este proceso ágilmente mediante la función: `step_normalize()`.

```
ames %>% select(Sale_Price, Neighborhood, Gr_Liv_Area, Year_Built, Bldg_Type) %>%
  head(5)
```

```
## # A tibble: 5 x 5
##   Sale_Price Neighborhood Gr_Liv_Area Year_Built Bldg_Type
##        <int> <fct>              <int>      <int> <fct>
## 1     215000 North_Ames          1656       1960 OneFam
## 2     105000 North_Ames           896       1961 OneFam
## 3     172000 North_Ames          1329       1958 OneFam
## 4     244000 North_Ames          2110       1968 OneFam
## 5     189900 Gilbert             1629       1997 OneFam
```

```
simple_ames <- recipe(Sale_Price ~ ., data = ames) %>%
  step_normalize(all_numeric_predictors())

simple_ames
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         73
##
## Operations:
##
## Centering and scaling for all_numeric_predictors()
```

```
simple_ames %>%
  prep() %>%
  bake(new_data = NULL) %>%
  select(Sale_Price, Neighborhood, Gr_Liv_Area, Year_Built, Bldg_Type) %>%
  head(5)
```

```
## # A tibble: 5 x 5
##   Sale_Price Neighborhood Gr_Liv_Area Year_Built Bldg_Type
##        <int> <fct>              <dbl>      <dbl> <fct>
## 1     215000 North_Ames         0.309     -0.375 OneFam
## 2     105000 North_Ames        -1.19      -0.342 OneFam
## 3     172000 North_Ames        -0.338     -0.442 OneFam
## 4     244000 North_Ames         1.21      -0.111 OneFam
## 5     189900 Gilbert            0.256      0.848 OneFam
```

2.6.1.2 Dicotomización de categorías

Otra transformación necesaria en la mayoría de los modelos predictivos en la creación de las variables dummy. Se mencionó anteriormente que los modelos requieren de una matriz numérica de características explicativas que permita calcular patrones estadísticos para predecir la variable de respuesta. El proceso de dicotomización consiste en **crear una variable dicotómica por cada categoría de una columna con valores nominales**.

```
ames %>% select(Sale_Price, Bldg_Type) %>% head(5)
```

```
## # A tibble: 5 x 2
##   Sale_Price Bldg_Type
##        <int> <fct>
## 1     215000 OneFam
## 2     105000 OneFam
## 3     172000 OneFam
```

```
## 4     244000 OneFam
## 5     189900 OneFam
```

```
ames %>% select(Bldg_Type) %>% distinct() %>% pull()
```

```
## [1] OneFam   TwnhsE   Twnhs    Duplex   TwoFmCon
## Levels: OneFam TwoFmCon Duplex Twnhs TwnhsE
```

```
simple_ames <- recipe(Sale_Price ~ Bldg_Type, data = ames) %>%
  step_dummy(all_nominal_predictors()) %>%
  prep()

simple_ames
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          1
##
## Training data contained 2930 data points and no missing data.
##
## Operations:
##
## Dummy variables from Bldg_Type [trained]
```

```
simple_ames %>% bake(new_data = NULL) %>% head(5)
```

```
## # A tibble: 5 x 5
##   Sale_Price Bldg_Type_TwoFmC~ Bldg_Type_Duplex Bldg_Type_Twnhs Bldg_Type_TwnhsE
##        <int>             <dbl>            <dbl>           <dbl>            <dbl>
## 1     215000                 0                0               0                0
## 2     105000                 0                0               0                0
## 3     172000                 0                0               0                0
## 4     244000                 0                0               0                0
## 5     189900                 0                0               0                0
```

El proceso de dicotomización demanda que únicamente `(n-1)` categorías sean expresadas, mientras que la restante será considerada la **categoría default o basal**. Esta última categoría es la usada en el modelo cuando todas las demás se encuentran ausentes.

2.6.1.3 Codificación de datos cualitativos nuevos o faltantes

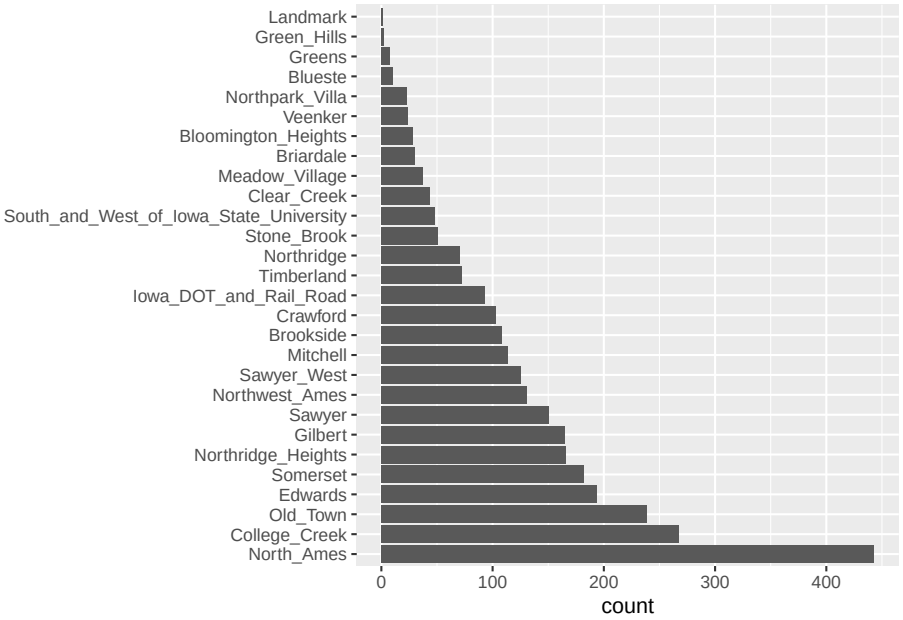
Una de las tareas de ingeniería de datos más comunes es el tratamiento de datos faltantes, datos no antes vistos y datos con poca frecuencia. **El problema principal con estos casos es que los modelos no saben cómo relacionar estos eventos con futuras predicciones**. Es conveniente realizar las transformaciones necesarias de tratamiento de estos datos antes de pasar a la etapa de modelado.

Por ejemplo:

- `step_unknown()` cambia los valores perdidos en un nivel de factor "desconocido".
- `step_other()` analiza las frecuencias de los niveles de los factores en el conjunto de datos y convierte los valores que ocurren con poca frecuencia a un nivel general de "otro", con un umbral que se puede especificar.
- `step_novel()` puede asignar un nuevo nivel si anticipamos que se puede encontrar un nuevo factor en datos futuros.

Un buen ejemplo es el predictor de **vecindad** en nuestros datos. Aquí hay dos vecindarios que tienen menos de cinco propiedades.

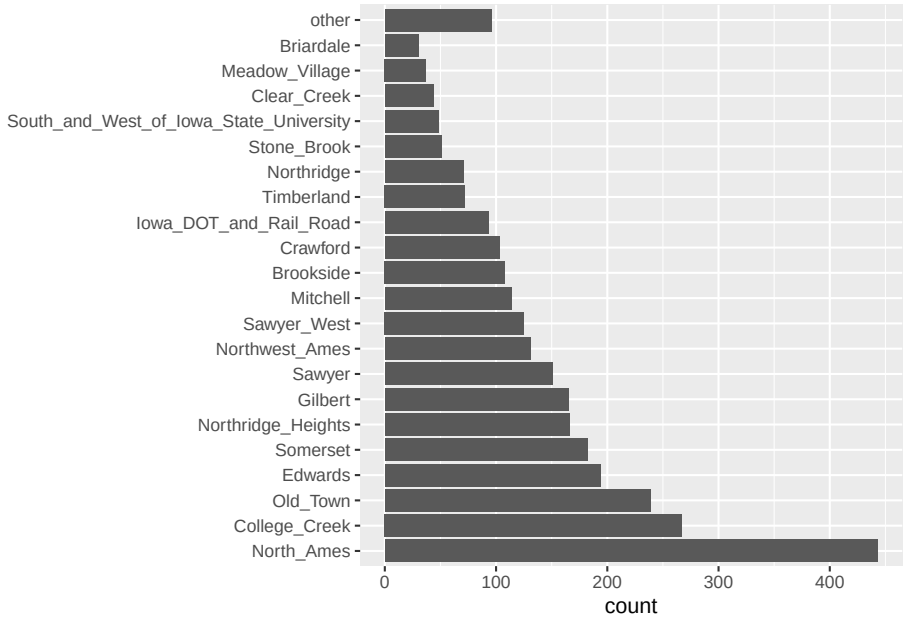
```
ggplot(data = ames, aes(y = Neighborhood)) +
  geom_bar() +
  labs(y = NULL)
```



Para algunos modelos, puede resultar problemático tener variables dummy con una sola entrada distinta de cero en la columna. Como mínimo, es muy improbable que estas características sean importantes para un modelo.

Si agregamos `step_other (Neighborhood, threshold = 0.01)` a nuestra receta, el último 1% de los vecindarios se agrupará en un nuevo nivel llamado "otro", esto atrapará a 8 vecindarios.

```
simple_ames <- recipe(
  Sale_Price ~ Neighborhood + Gr_Liv_Area + Year_Built + Bldg_Type,
  data = ames) %>%
  step_other(Neighborhood, threshold = 0.01) %>%
  prep()

ejemplo <- juice(simple_ames)

ggplot(ejemplo, aes(y = Neighborhood)) +
  geom_bar() +
  labs(y = NULL)
```



2.6.2 Imputaciones

La función `step_unknown` crea una categoría nombrada `unknown`, la cual sirve como reemplazo de datos categóricos faltantes, sin embargo, para imputar datos

numéricos se requiere de otra estrategia. Las imputaciones o sustituciones más comunes son realizadas a través de medidas de tendencia central tales como la media y mediana. A continuación se muestra un ejemplo:

```
ames_na <- ames
ames_na[sample(nrow(ames), 5), c("Gr_Liv_Area", "Lot_Area")] <- NA

ames_na %>% filter(is.na(Gr_Liv_Area) | is.na(Lot_Area)) %>%
  select(Sale_Price, Gr_Liv_Area, Lot_Area)
```

```
## # A tibble: 5 x 3
##   Sale_Price Gr_Liv_Area Lot_Area
##        <int>       <int>    <int>
## 1     153000          NA       NA
## 2     180000          NA       NA
## 3     170000          NA       NA
## 4     215000          NA       NA
## 5     157000          NA       NA
```

```
simple_ames <- recipe(Sale_Price ~ Gr_Liv_Area + Lot_Area, data = ames_na) %>%
  step_impute_mean(Gr_Liv_Area) %>%
  step_impute_median(Lot_Area) %>%
  prep()

bake(simple_ames, new_data = ames_na) %>%
  filter(is.na(Gr_Liv_Area) | is.na(Lot_Area))
```

```
## # A tibble: 0 x 3
## # ... with 3 variables: Gr_Liv_Area <int>, Lot_Area <int>, Sale_Price <int>
```

Forzamos algunos renglones a que sean omitidos aleatoriamente. Posteriormente, estos valores son imputados mediante su media y mediana.

2.6.3 Agregar o modificar columnas

Quizá la transformación más usada sea la agregación o mutación de columnas existentes. Similar a la función `mutate()` de *dplyr*, la función `step_mutate()` se encarga de realizar esta tarea dentro de un pipeline o receta.

```
ejemplo <- recipe(
  Sale_Price ~ Neighborhood + Gr_Liv_Area + Year_Built + Bldg_Type + Year_Remod_Add,
  data = ames) %>%
  step_mutate(
```

```
    Sale_Price_Peso = Sale_Price * 19.87,
    Last_Inversion = Year_Remod_Add - Year_Built
    ) %>%
  step_arrange(desc(Last_Inversion)) %>%
  prep()

ejemplo
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          5
##
## Training data contained 2930 data points and no missing data.
##
## Operations:
##
## Variable mutation for ~Sale_Price * 19.87, ~Year_Remod_Add - Yea... [trained]
## Row arrangement using ~desc(Last_Inversion) [trained]
```

```
ejemplo %>% bake(new_data = NULL) %>%
  select(Sale_Price, Sale_Price_Peso, Year_Remod_Add, Year_Built, Last_Inversion)
```

```
## # A tibble: 2,930 x 5
##    Sale_Price Sale_Price_Peso Year_Remod_Add Year_Built Last_Inversion
##         <int>           <dbl>          <int>      <int>          <int>
##  1     131000        2602970           2007       1880            127
##  2     265979        5285003.          2003       1880            123
##  3     295000        5861650           2002       1880            122
##  4      94000        1867780           1996       1875            121
##  5     138000        2742060           2006       1890            116
##  6     122000        2424140           1987       1872            115
##  7     240000        4768800           2002       1890            112
##  8     119600        2376452           2006       1895            111
##  9     124000        2463880           1991       1880            111
## 10     100000        1987000           1995       1885            110
## # ... with 2,920 more rows
```

En este ejemplo se realiza la creación de una nueva variable y la modificación de una ya existente.

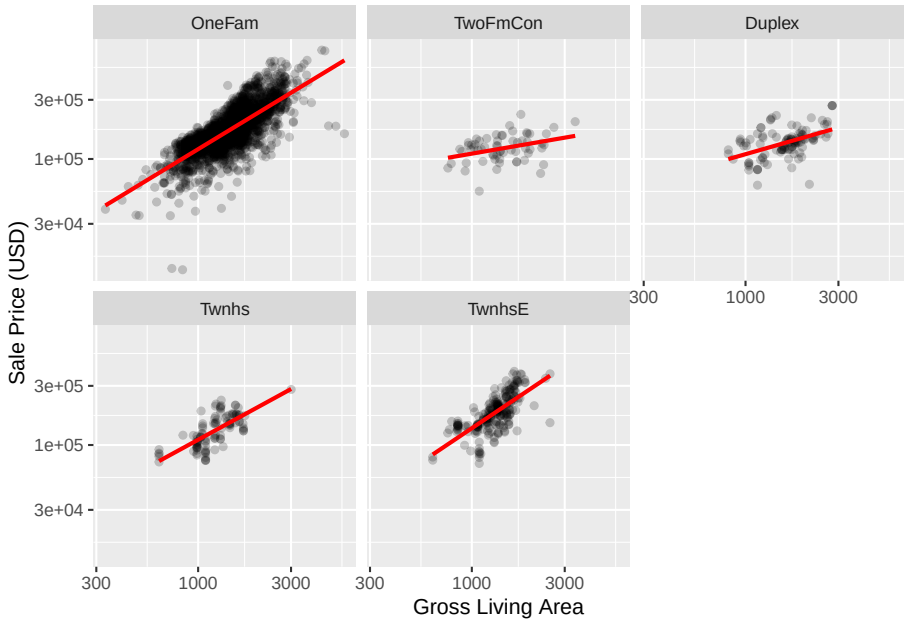
2.6.4 Interacciones

Los efectos de interacción involucran dos o más predictores. Tal efecto **ocurre cuando un predictor tiene un efecto sobre el resultado que depende de uno o más predictores.**

Numéricamente, un término de interacción entre predictores se codifica como su producto. Las interacciones solo se definen en términos de su efecto sobre el resultado y pueden ser combinaciones de diferentes tipos de datos (por ejemplo, numéricos, categóricos, etc.).

Después de explorar el conjunto de datos de Ames, podríamos encontrar que las pendientes de regresión para el área habitable bruta difieren para los diferentes tipos de edificios:

```
ggplot(ames, aes(x = Gr_Liv_Area, y = Sale_Price)) +
  geom_point(alpha = .2) +
  facet_wrap(~ Bldg_Type) +
  geom_smooth(method = lm, formula = y ~ x, se = FALSE, col = "red") +
  scale_x_log10() +
  scale_y_log10() +
  labs(x = "Gross Living Area", y = "Sale Price (USD)")
```



Con la receta actual, `step_dummy()` ya ha creado variables ficticias. ¿Cómo combinaríamos estos para una interacción? El paso adicional se vería como `step_interact(~ términos de interacción)` donde los términos en el lado

derecho de la tilde son las interacciones. Estos pueden incluir selectores, por lo que sería apropiado usar:

```
simple_ames <- recipe(Sale_Price ~ Neighborhood + Gr_Liv_Area + Year_Built + Bldg_Type
         data = ames) %>%
  step_other(Neighborhood, threshold = 0.05) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_interact( ~ Gr_Liv_Area:starts_with("Bldg_Type_") ) %>%
  prep()

simple_ames %>% bake(new_data = NULL) %>% glimpse()
```

```
## Rows: 2,930
## Columns: 19
## $ Gr_Liv_Area                      <int> 1656, 896, 1329, 2110, 1629, 1604, 13~
## $ Year_Built                       <int> 1960, 1961, 1958, 1968, 1997, 1998, 2~
## $ Sale_Price                       <int> 215000, 105000, 172000, 244000, 18990~
## $ Neighborhood_College_Creek       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Neighborhood_Old_Town            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Neighborhood_Edwards             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Neighborhood_Somerset            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Neighborhood_Northridge_Heights  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Neighborhood_Gilbert             <dbl> 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1~
## $ Neighborhood_Sawyer              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Neighborhood_other               <dbl> 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0~
## $ Bldg_Type_TwoFmCon               <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Bldg_Type_Duplex                 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Bldg_Type_Twnhs                  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Bldg_Type_TwnhsE                 <dbl> 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0~
## $ Gr_Liv_Area_x_Bldg_Type_TwoFmCon <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Gr_Liv_Area_x_Bldg_Type_Duplex   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Gr_Liv_Area_x_Bldg_Type_Twnhs    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Gr_Liv_Area_x_Bldg_Type_TwnhsE   <dbl> 0, 0, 0, 0, 0, 0, 1338, 1280, 1616, 0~
```

Se pueden especificar interacciones adicionales en esta fórmula separándolas con el signo $*$.

2.6.5 Transformaciones generales

Reflejando las operaciones originales de dplyr, los siguientes pasos se pueden usar para realizar una variedad de operaciones básicas a los datos.

- `step_select()`: Selecciona un subconjunto de variables específicas en el conjunto de datos.

- `step_mutate()`: Crea una nueva variable o modifica una existente usando `dplyr::mutate()`.
- `step_mutate_at()`: Lee una especificación de un paso de receta que modificará las variables seleccionadas usando una función común a través de `dplyr::mutate_at()`.
- `step_filter()`: Crea una especificación de un paso de receta que eliminará filas usando `dplyr::filter()`.
- `step_arrange()`: Ordena el conjunto de datos de acuerdo con una o más variables.
- `step_rm()`: Crea una especificación de un paso de receta que eliminará las variables según su nombre, tipo o función.
- `step_nzv()`: Realiza una selección de variables eliminando todas aquellas cuya varianza se encuentre cercana a cero.
- `step_naomit()`: Elimina todos los renglones que tengan alguna variable con valores perdidos.
- `step_normalize()`: Centra y escala las variables numéricas especificadas, generando una transformación a una distribución normal estándar.
- `step_range()`: Transforma el rango de un conjunto de variables numéricas al especificado.
- `step_interact()`: Crea un nuevo conjunto de variables basadas en la interacción entre dos variables.
- `step_ratio()`: Crea una nueva variable a partir del cociente entre dos variables.
- `all_predictors()`: Selecciona a todos los predictores del conjunto de entrenamineto para aplicarles alguna de las funciones mencionadas.
- `all_numeric_predictors()`: Selecciona a todos los predictores numéricos del conjunto de entrenamineto para aplicarles alguna de las funciones mencionadas.
- `all_nominal_predictors()`: Selecciona a todos los predictores nominales del conjunto de entrenamineto para aplicarles alguna de las funciones mencionadas.

La guía completa de las familia de funciones *step* puede consultarse en la documentación oficial

2.7 Partición de datos

Cuando hay una gran cantidad de datos disponibles, una estrategia inteligente es asignar subconjuntos específicos de datos para diferentes tareas, en lugar de asignar la mayor cantidad posible solo a la estimación de los parámetros del modelo.

Si el conjunto inicial de datos no es lo suficientemente grande, habrá cierta superposición de cómo y cuándo se asignan nuestros datos, y es importante contar con una metodología sólida para la partición de datos.

2.7.1 Métodos comunes para particionar datos

El enfoque principal para la validación del modelo es dividir el conjunto de datos existente en dos conjuntos distintos:

- **Entrenamiento:** Este conjunto suele contener la mayoría de los datos, los cuales sirven para la construcción de modelos donde se pueden ajustar diferentes modelos, se investigan estrategias de ingeniería de características, etc.

  La mayor parte del proceso de modelado se utiliza este conjunto.

- **Prueba:** La otra parte de las observaciones se coloca en este conjunto. Estos datos se mantienen en reserva hasta que se elijan uno o dos modelos como los de mejor rendimiento.

  El conjunto de prueba se utiliza como árbitro final para determinar la eficiencia del modelo, por lo que es fundamental mirar el conjunto de prueba una sola vez.

Supongamos que asignamos el 80% de los datos al conjunto de entrenamiento y el 20% restante a las pruebas. El método más común es utilizar un muestreo

aleatorio simple. El paquete *rsample* tiene herramientas para realizar divisiones de datos como esta; la función `initial_split()` fue creada para este propósito.

```
library(tidymodels)

tidymodels_prefer()

# Fijar un número aleatorio con para que los resultados puedan ser reproducibles
set.seed(123)

# Partición 80/20 de los datos
ames_split <- initial_split(ames, prop = 0.80)
ames_split
```

```
## <Analysis/Assess/Total>
## <2344/586/2930>
```

La información impresa denota la cantidad de datos en el conjunto de entrenamiento ($n = 2,344$), la cantidad en el conjunto de prueba ($n = 586$) y el tamaño del grupo original de muestras ($n = 2,930$).

El objeto `ames_split` es un objeto *rsplit* y solo contiene la información de partición; para obtener los conjuntos de datos resultantes, aplicamos dos funciones más:

```
ames_train <- training(ames_split)
ames_test  <-  testing(ames_split)

dim(ames_train)
```

```
## [1] 2344   74
```

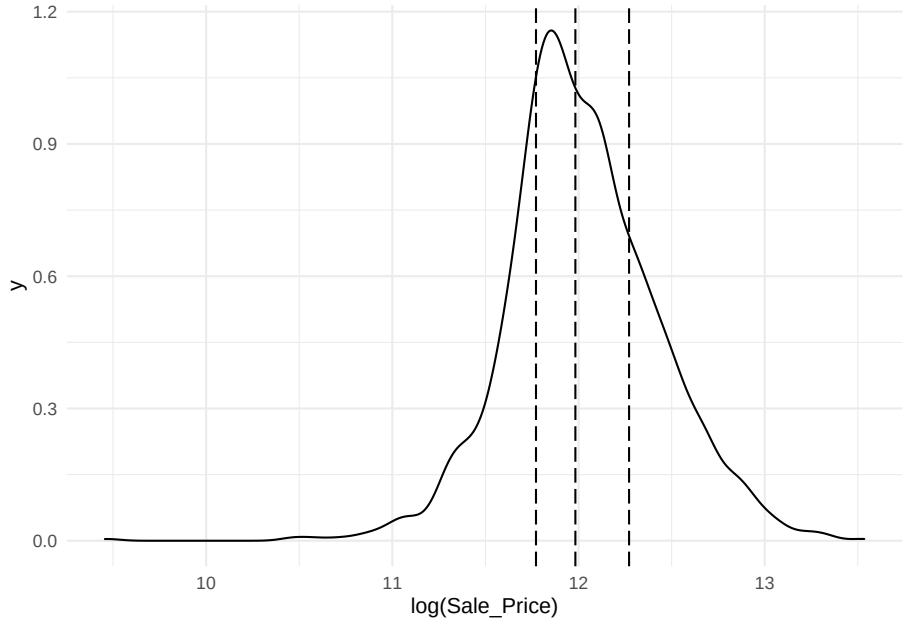
El muestreo aleatorio simple es apropiado en muchos casos, pero hay excepciones.

Cuando hay un desbalance de clases en los problemas de clasificación, el uso de una muestra aleatoria simple puede asignar al azar estas muestras poco frecuentes de manera desproporcionada al conjunto de entrenamiento o prueba.

Para evitar esto, se puede utilizar un muestreo estratificado. La división de entrenamiento/prueba se lleva a cabo por separado dentro de cada clase y luego estas submuestras se combinan en el conjunto general de entrenamiento y prueba.

Para los problemas de regresión, los datos de los resultados se pueden agrupar artificialmente en cuartiles y luego realizar un muestreo estratificado cuatro

veces por separado. Este es un método eficaz para mantener similares las distribuciones del resultado entre el conjunto de entrenamiento y prueba.



Observamos que la distribución del precio de venta está sesgada a la derecha. Las casas más caras no estarían bien representadas en el conjunto de entrenamiento con una simple partición; esto aumentaría el riesgo de que nuestro modelo sea ineficaz para predecir el precio de dichas propiedades.

Las líneas verticales punteadas indican los cuatro cuartiles para estos datos. Una muestra aleatoria estratificada llevaría a cabo la división 80/20 dentro de cada uno de estos subconjuntos de datos y luego combinaría los resultados. En *rsample*, esto se logra usando el argumento de estratos:

```
set.seed(123)
ames_split <- initial_split(ames, prop = 0.80, strata = Sale_Price)
ames_train <- training(ames_split)
ames_test  <-  testing(ames_split)
```

Hay muy pocas desventajas en el uso de muestreo estratificado.

Un caso es cuando los datos tienen un componente de tiempo, como los datos de series de tiempo. Aquí, es más común utilizar los datos más recientes como conjunto de prueba.

El paquete *rsample* contiene una función llamada `initial_time_split()` que es muy similar a `initial_split()`. En lugar de usar un muestreo aleatorio, el argumento `prop` denota qué proporción de la primera parte de los datos debe

usarse como conjunto de entrenamiento; la función asume que los datos se han clasificado previamente en un orden apropiado.

2.7.2 ¿Qué proporción debería ser usada?

No hay un porcentaje de división óptimo para el conjunto de entrenamiento y prueba. Muy pocos datos en el conjunto de entrenamiento obstaculizan la capacidad del modelo para encontrar estimaciones de parámetros adecuadas y muy pocos datos en el conjunto de prueba reducen la calidad de las estimaciones de rendimiento.

Se debe elegir un porcentaje que cumpla con los objetivos de nuestro proyecto con consideraciones que incluyen:

- Costo computacional en el entrenamiento del modelo.
- Costo computacional en la evaluación del modelo.
- Representatividad del conjunto de formación.
- Representatividad del conjunto de pruebas.

Los porcentajes de división más comunes comunes son:

- Entrenamiento: 80%, Prueba: 20%
- Entrenamiento: 67%, Prueba: 33%
- Entrenamiento: 50%, Prueba: 50%

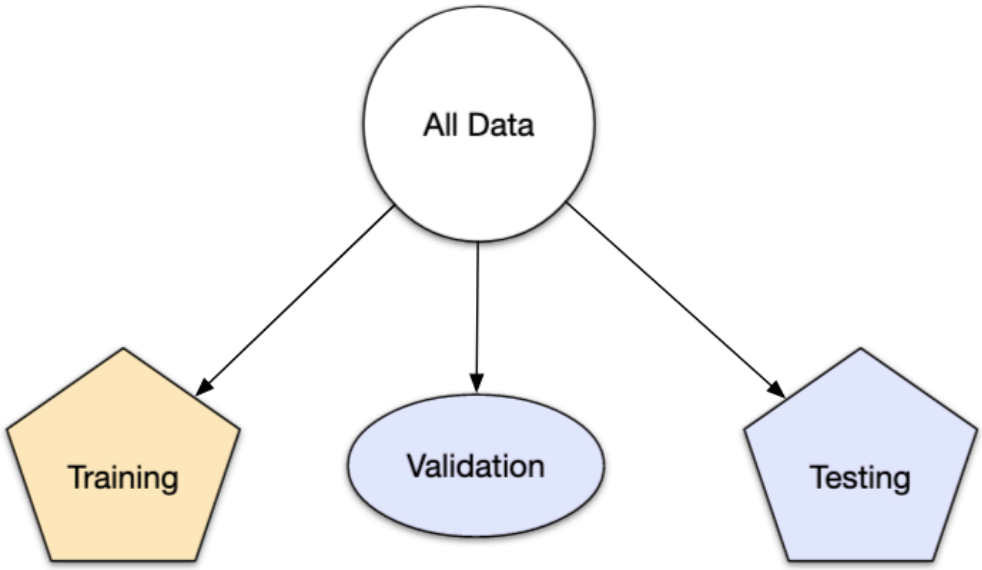
2.7.3 Conjunto de validación

El conjunto de validación se definió originalmente cuando los investigadores se dieron cuenta de que medir el rendimiento del conjunto de entrenamiento conducía a resultados que eran demasiado optimistas.

Esto llevó a modelos que se sobreajustaban, lo que significa que se desempeñaron muy bien en el conjunto de entrenamiento pero mal en el conjunto de prueba.

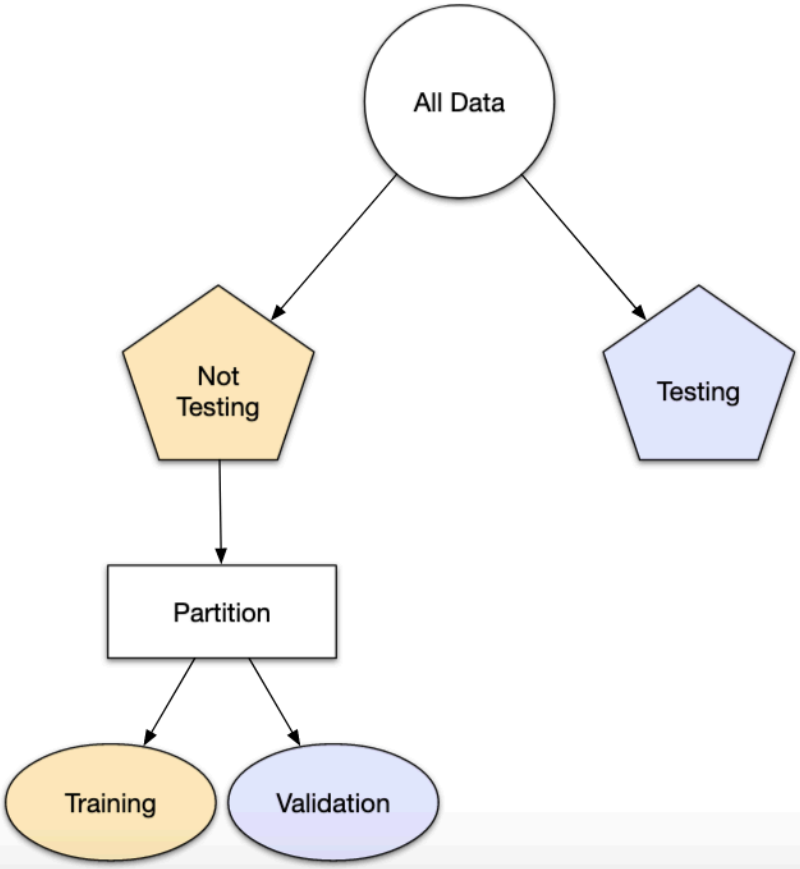
Para combatir este problema, se retuvo un pequeño conjunto de datos de *validación* y se utilizó para medir el rendimiento del modelo mientras este está siendo entrenado. Una vez que la tasa de error del conjunto de validación comenzara a aumentar, la capacitación se detendría.

En otras palabras, el conjunto de validación es un medio para tener una idea aproximada de qué tan bien se desempeñó el modelo antes del conjunto de prueba.



Los conjuntos de validación se utilizan a menudo cuando el conjunto de datos original es muy grande. En este caso, una sola partición grande puede ser adecuada para caracterizar el rendimiento del modelo sin tener que realizar múltiples iteraciones de remuestreo.

Con *rsample*, un conjunto de validación es como cualquier otro objeto de remuestreo; este tipo es diferente solo en que tiene una sola iteración




```
set.seed(12)
val_set <- validation_split(ames_train, prop = 3/4, strata = NULL)
val_set #val_set contiene el conjunto de entrenamiento y validación.
```

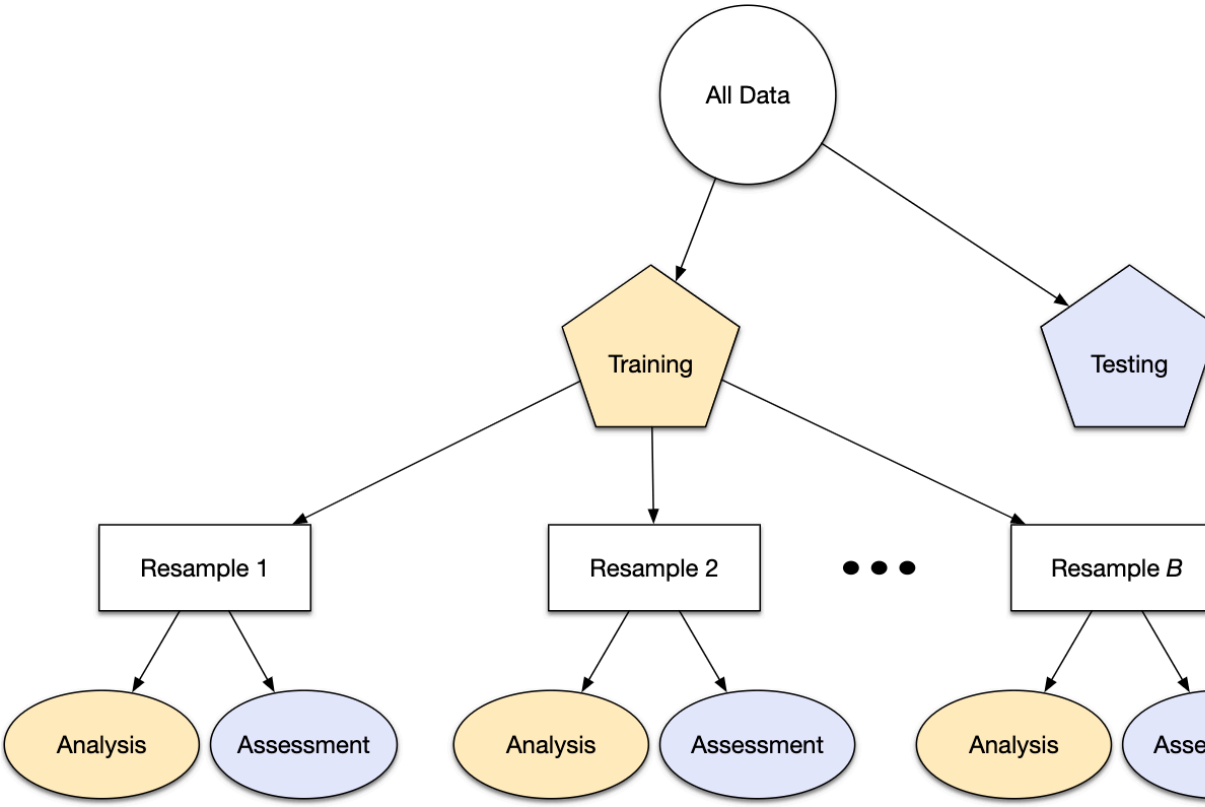
```
## # Validation Set Split (0.75/0.25)
## # A tibble: 1 x 2
##   splits             id
##   <list>             <chr>
## 1 <split [1756/586]> validation
```

Esta función regresa una columna para los objetos de división de datos y una columna llamada id que tiene una cadena de caracteres con el identificador de remuestreo.

El argumento de estratos hace que el muestreo aleatorio se lleve a cabo dentro de la variable de estratificación. Esto puede ayudar a garantizar que el número

de datos en los datos del análisis sea equivalente a las proporciones del conjunto de datos original. (Los estratos inferiores al 10% del total se agrupan).

Otra opción de muestreo bastante común es la realizada mediante múltiples submuestras de los datos originales.



Diversos métodos se revisarán a lo largo del curso.

2.7.4 Leave-one-out cross-validation

La validación cruzada es una manera de predecir el ajuste de un modelo a un hipotético conjunto de datos de prueba cuando no disponemos del conjunto explícito de datos de prueba.

El método $LOOCV$ en un método iterativo que se inicia empleando como conjunto de entrenamiento todas las observaciones disponibles excepto una, que se excluye para emplearla como validación.

Si se emplea una única observación para calcular el error, este varía mucho dependiendo de qué observación se haya seleccionado. Para evitarlo, el proceso se repite tantas veces como observaciones disponibles se tengan, excluyendo en cada iteración una observación distinta, ajustando el modelo con el resto y calculando el error con dicha observación.

Finalmente, el error estimado por el es el promedio de todos lo i errores calculados.

La principal desventaja de este método es su costo computacional. El proceso requiere que el modelo sea reajustado y validado tantas veces como observaciones disponibles se tengan lo que en algunos casos puede ser muy complicado.

rsample contiene la función `loo_cv()`.

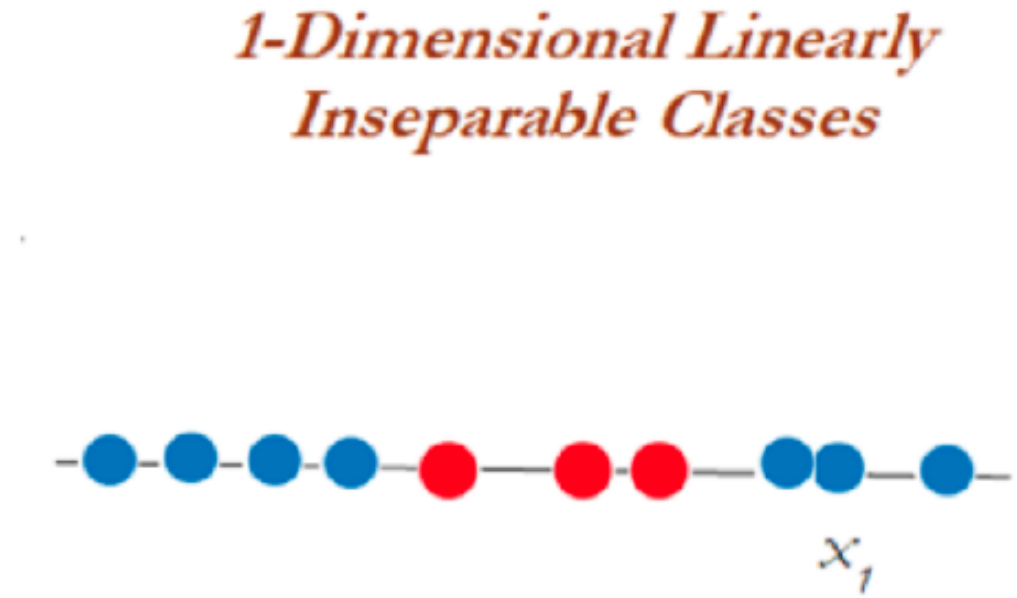
```
set.seed(55)
ames_loo <- loo_cv(ames_train)
ames_loo
```

```
## # Leave-one-out cross-validation
## # A tibble: 2,342 x 2
##    splits           id
##    <list>           <chr>
##  1 <split [2341/1]> Resample1
##  2 <split [2341/1]> Resample2
##  3 <split [2341/1]> Resample3
##  4 <split [2341/1]> Resample4
##  5 <split [2341/1]> Resample5
##  6 <split [2341/1]> Resample6
##  7 <split [2341/1]> Resample7
##  8 <split [2341/1]> Resample8
##  9 <split [2341/1]> Resample9
## 10 <split [2341/1]> Resample10
## # ... with 2,332 more rows
```

Chapter 3

Support Vector Machine (SVM)

Support vector machine, llamadas SVM, son un algoritmo de aprendizaje supervisado que se puede utilizar para problemas de clasificación y regresión. Se utiliza para conjuntos de datos más pequeños, ya que tarda demasiado en procesarse.



El principal objetivo de esta técnica es encontrar el **Hiperplano de Separación Óptima**, también conocido como *Boundary Decision*, el cual separa a las clases involucradas.

Para entender este algoritmo es necesario entender 3 conceptos principales:

1. Maximum margin classifiers
2. Support vector classifiers
3. Support vector machines

Estudiemos cada uno de estos principios.

3.1 Maximum Margin Classifier

A menudo se generalizan con máquinas de vectores de soporte, pero SVM tiene muchos más parámetros en comparación. El *clasificador de margen máximo* considera un hiperplano con ancho de separación máxima para clasificar los datos. Sin embargo, se pueden dibujar infinitos hiperplanos en un conjunto de datos por lo que es importante elegir el hiperplano ideal para la clasificación.

En un espacio *n-dimensional*, un hiperplano es un subespacio de la dimensión n-1. Es decir, si los datos tienen un espacio bidimensional, entonces el hiperplano puede ser una línea recta que divide el espacio de datos en dos mitades y pasa por la siguiente ecuacion:

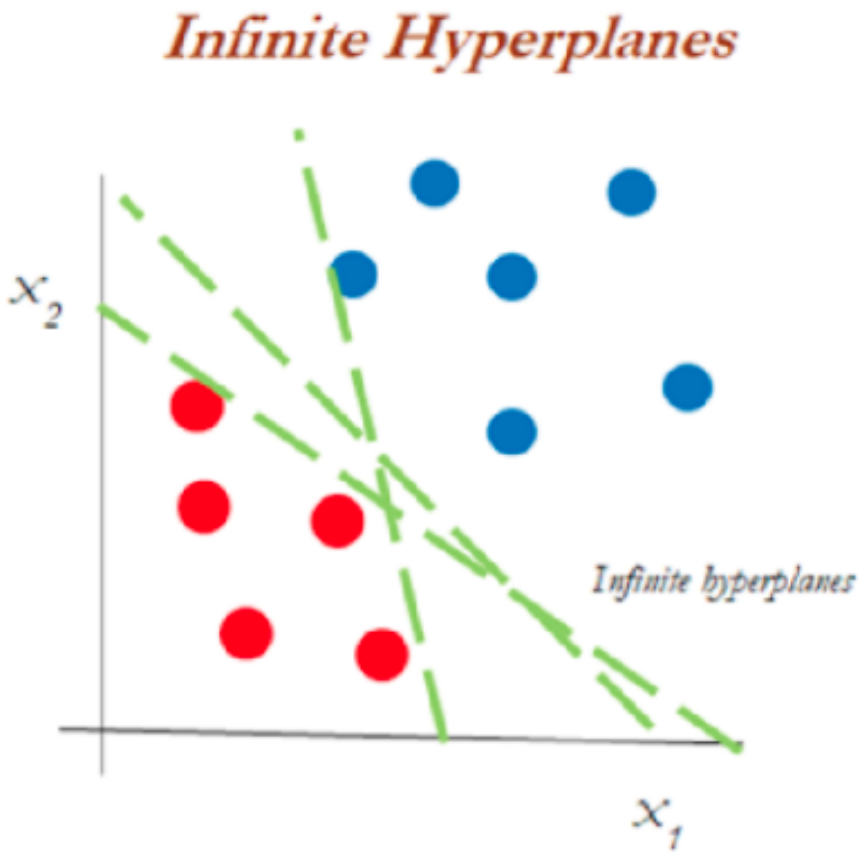
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

Las observaciones que caen en el hiperplano sigue la ecuación anterior. Las observaciones que caen en la región por encima o por debajo del hiperplano sigue las siguientes ecuaciones:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$$

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 < 0$$

El clasificador de margen máximo a menudo falla en la situación de casos no separables en los que no puede asignar un hiperplano diferente para clasificar datos no separables. Para tales casos, un clasificador de vectores de soporte viene al rescate.



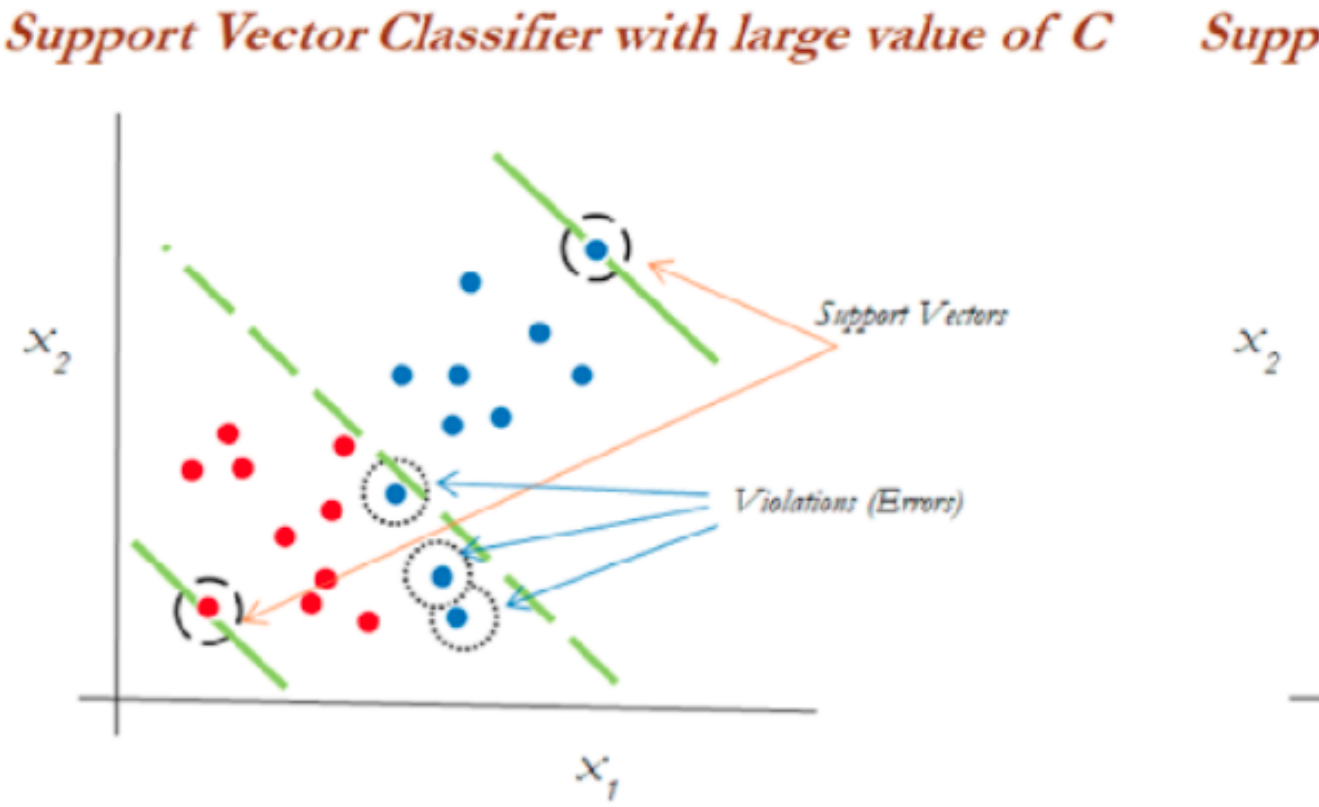
Del diagrama anterior, podemos suponer infinitos hiperplanos (izquierda). El clasificador de margen máximo viene con un solo hiperplano que divide los datos como en la gráfica de la derecha. **Los datos que tocan los hiperplanos positivo y negativo se denominan vectores de soporte**.

3.2 Support Vector Classifiers

Los vectores de soporte son las observaciones que están más cerca del hiperplano e influyen en la posición y orientación del hiperplano. Este tipo de clasificador puede considerarse como una versión extendida del clasificador de margen máximo. Cuando tratamos con datos de la vida real, encontramos que la mayoría de las observaciones están en clases superpuestas. Es por eso que se implementan clasificadores de vectores de soporte.

Usando estos vectores de soporte, maximizamos el margen del clasificador. Eliminar los vectores de soporte cambiará la posición del hiperplano. Estos son los

puntos que nos ayudan a construir nuestro *SVM*. Consideremos un **parámetro de ajuste C**. En este clasificador, el alto valor de C puede darnos un modelo robusto. Un valor más bajo de C nos da un modelo flexible. Entendamos con el siguiente diagrama.



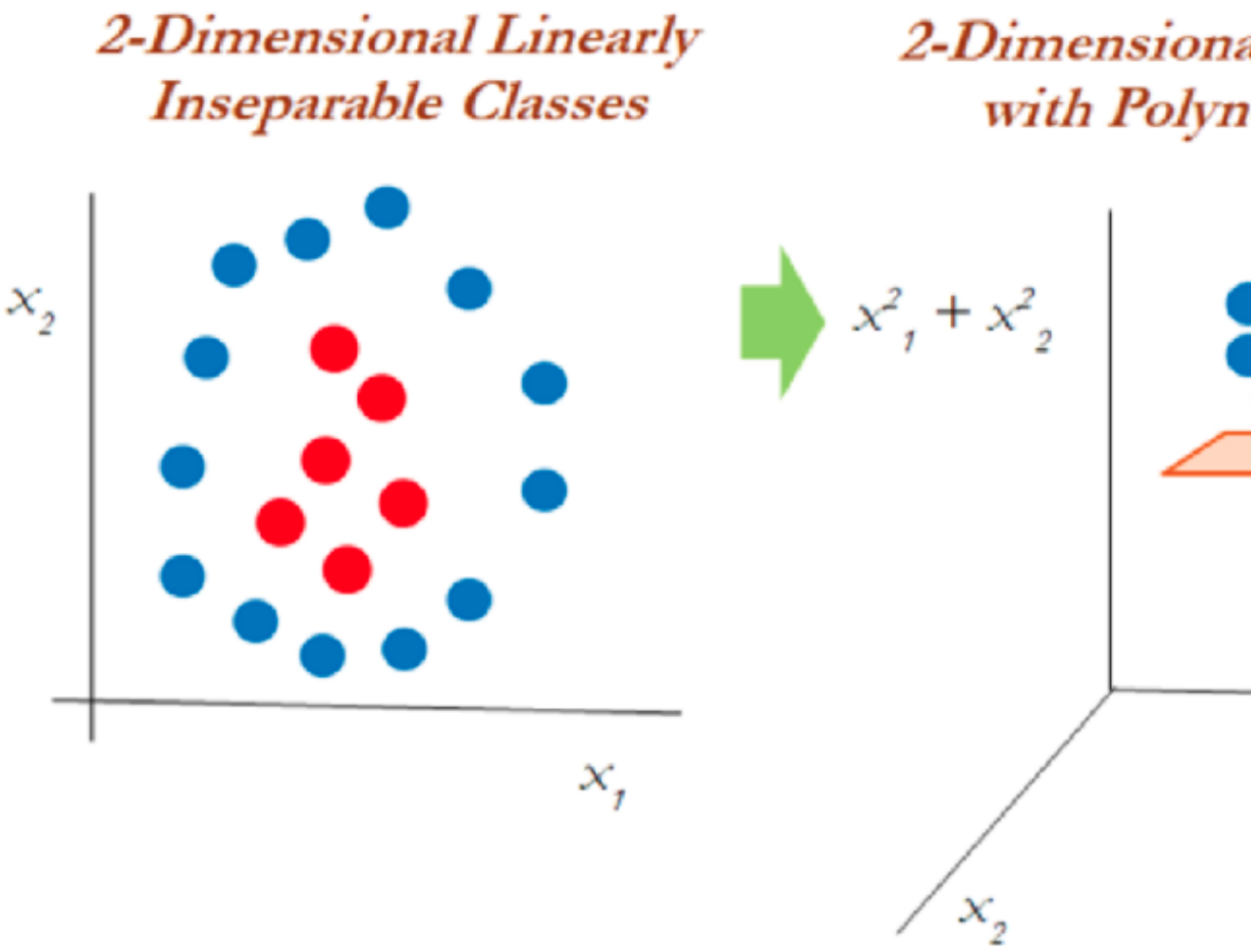
Podemos ver en el gráfico de la izquierda que los valores más altos de C generaron más errores que se consideran una **violación o infracción**. El diagrama de la derecha muestra un valor más bajo de C y no brinda suficientes posibilidades de infracción al reducir el ancho del margen.

3.3 Support Vector Machine

El enfoque de la máquina de vectores de soporte se considera durante una decisión no lineal y los datos no son separables por un clasificador de vectores de soporte, independientemente de la función de costo.

Cuando es casi imposible separar clases de manera no lineal, aplicamos el truco llamado **truco del kernel** el cual ayuda a manejar la separación de los datos.

En el gráfico anterior, los datos que eran inseparables en una dimensión se separaron una vez que se transformaron a un espacio de dos dimensiones después de aplicar una **transformación mediante kernel polinomial de segundo grado**. Ahora veamos cómo manejar los datos bidimensionales linealmente inseparables.

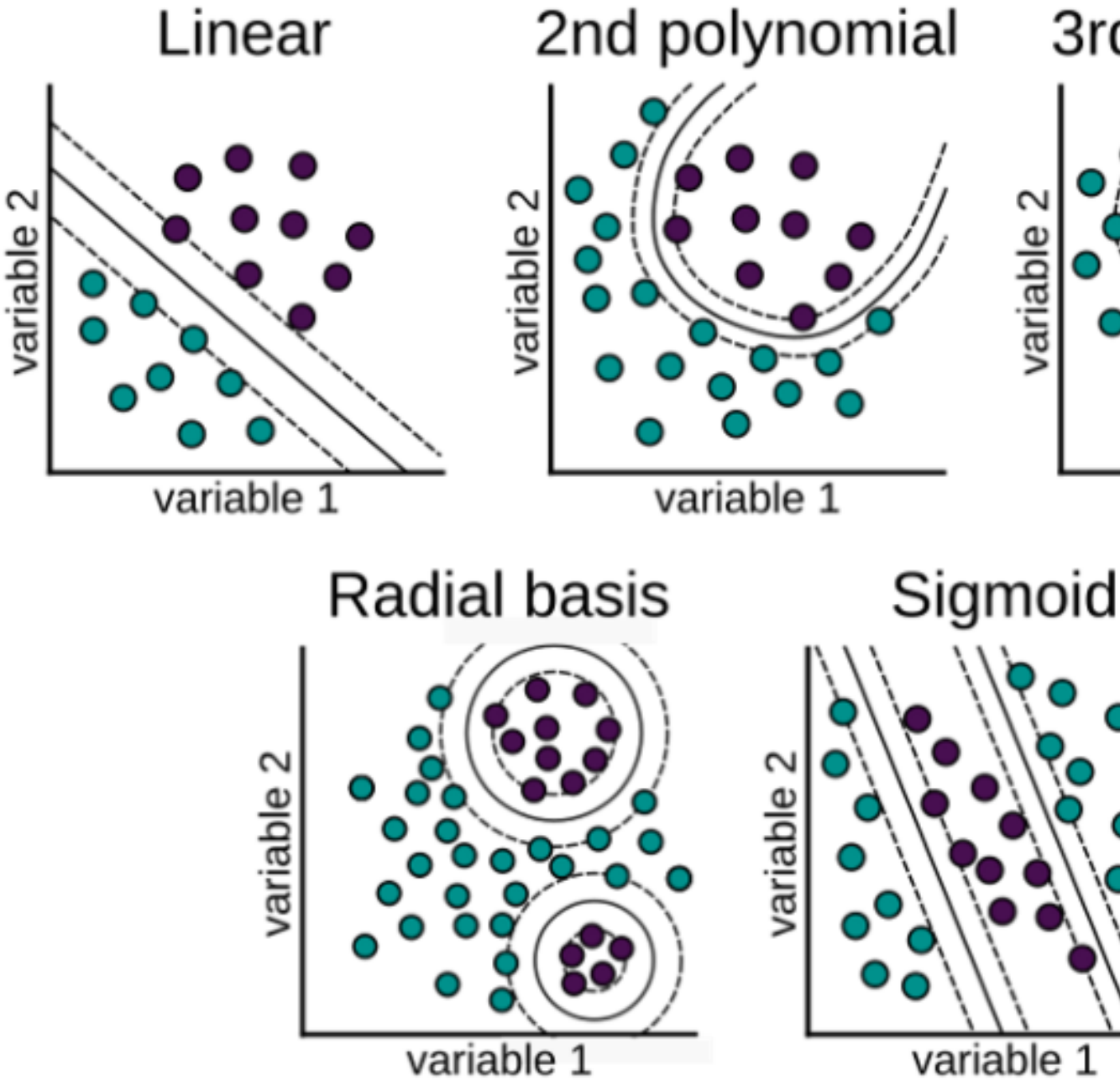


En datos bidimensionales, el núcleo polinomial de segundo grado se aplica utilizando un plano lineal después de transformarlo a dimensiones superiores.

3.4 El truco del Kernel

Las funciones Kernel son métodos con los que se utilizan clasificadores lineales como *SVM* para clasificar puntos de datos separables no linealmente. Esto se hace representando los puntos de datos en un espacio de mayor dimensión que su original. Por ejemplo, los datos 1D se pueden representar como datos 2D en el espacio, los datos 2D se pueden representar como datos 3D, etcétera.

El truco del kernel ofrece una **forma de calcular las relaciones entre los puntos de datos** utilizando funciones del kernel y representar los datos de una manera más eficiente con menos cómputo. Los modelos que utilizan esta técnica se denominan **"modelos kernelizados"**.



Hay varias funciones que utiliza SVM para realizar esta tarea. Algunos de los más comunes son:

1. **El núcleo lineal:** Se utiliza para datos lineales. Esto simplemente representa los puntos de datos usando una relación lineal.

$$K(x, y) = (x^T \cdot y)$$

$$f(x) = w^T \cdot x + b$$

Esta formulación se presenta como solución al problema de optimización sobre w:

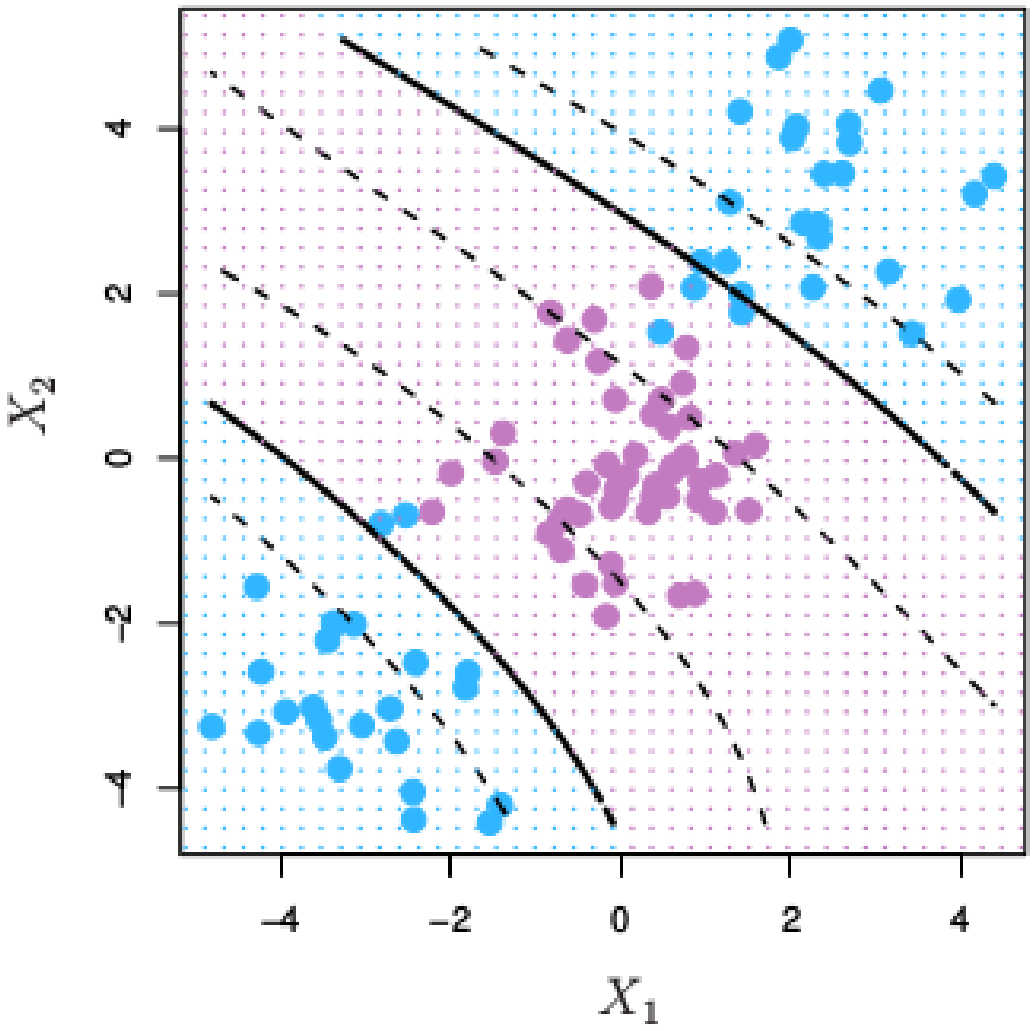
$$min_{w \in R^d} \parallel w \parallel^2 + C \sum_{i}^{N} max(0, 1 - y_i f(x_i))$$

$$s.a. \quad y_i(w^T x_i + b) \geq 1 - max(0, 1 - y_i f(x_i))$$

2. **Función de núcleo polinomial:** Transforma los puntos de datos mediante el **uso del producto escalar** y la transformación de los datos en una "dimensión n", n podría ser cualquier valor de 2, 3, etcétera, es decir, la transformación será un producto al cuadrado o superior. Por lo tanto, representar datos en un espacio de mayor dimensión utilizando los nuevos puntos transformados.

$$K(x, y) = (c + x^T \cdot y)^p$$

Cuando se emplea $p = 1$ y $c = 0$, el resultado es el mismo que el de un kernel lineal. Si $p > 1$, se generan límites de decisión no lineales, aumentando la no linealidad a medida que aumenta p. No suele ser recomendable emplear valores de p mayores 5 por problemas de **overfitting**.



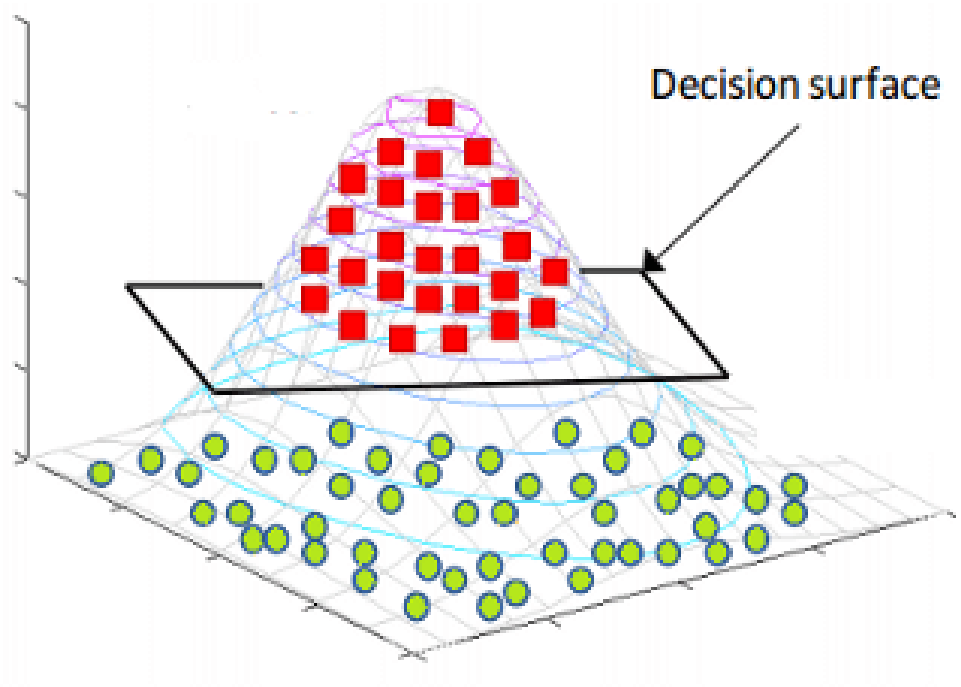
3. **La función de base radial (RBF):** Esta función se comporta como un "modelo de vecino más cercano ponderado". Transforma los datos representándolos en dimensiones infinitas,

La función Radial puede ser de Gauss o de Laplace. Esto depende de un hiperparámetro conocido como gamma γ. Cuanto menor sea el valor del hiperparámetro, menor será el sesgo y mayor la varianza. Mientras que un valor más alto de hiperparámetro da un sesgo más alto y menor varianza. Este es el núcleo más utilizado.

$$K(x, y) = exp(-\gamma \parallel x - y \parallel^2) = exp(-\frac{\parallel x - y \parallel^2}{2\sigma^2})$$

$$f(x) = w^T \cdot \phi(x) + b$$

Se realiza un mapeo de x a $\phi(x)$ en donde los datos son separables



Es recomendable probar el kernel **RBF**. Este kernel tiene dos ventajas: que solo tiene dos hiperparámetros que optimizar (γ y la penalización C común a todos los SVM) y que su flexibilidad puede ir desde un clasificador lineal a uno muy complejo.

4. **La función sigmoide:** También conocida como función tangente hiperbólica (Tanh), encuentra más aplicación en redes neuronales como función de activación. Esta función se utiliza en la clasificación de imágenes.

$$K(x, y) = tanh(\kappa x \cdot y - \delta)$$

¿Por qué se llama un "truco del kernel"? *SVM* vuelve a representar hábilmente los puntos de datos no lineales utilizando cualquiera de las funciones del kernel

de una manera que parece que los datos se han transformado, luego encuentra el hiperplano de separación óptimo. Sin embargo, en realidad, los puntos de datos siguen siendo los mismos, en realidad no se han transformado. Es por eso que se llama un 'truco del kernel'.

3.5 Ventajas y desventajas

Ventajas

- Es un modelo que ajusta bien con pocos datos
- Son flexibles en datos no estructurados, estructurados y semiestructurados.
- La función Kernel alivia las complejidades en casi cualquier tipo de datos.
- Se observa menos sobreajuste en comparación con otros modelos.

Desventajas

- El tiempo de entrenamiento es mayor cuando se calculan grandes conjuntos de datos.
- Los hiperparámetros suelen ser un desafío al interpretar su impacto.
- La interpretación general es difícil (black box).

3.6 Ajuste del modelo con R

Usaremos las recetas antes implementadas para ajustar tanto el modelo de regresión como el de clasificación. Exploraremos un conjunto de hiperparámetros para elegir el mejor modelo.

Recordemos que es importante separar los datos de entrenamiento y prueba, así como sub-particionar en fold a los datos de entrenamiento para realizar diferentes pruebas con distintas parametrizaciones de los modelos. Finalmente, calcularemos el error promedio y los mejores hiperparámetros a implementar.

```
library(tidymodels)

data(ames)

set.seed(4595)
ames_split <- initial_split(ames, prop = 0.75)
```

```
ames_train <- training(ames_split)
ames_test  <- testing(ames_split)
ames_folds<- vfold_cv(ames_train)
```

Contando con datos de entrenamiento, procedemos a realizar el feature engineering para extraer las mejores características que permitirán realizar las estimaciones en el modelo.

```
receta_casas <- recipe(Sale_Price ~ . , data = ames_train) %>%
  step_unknown(Alley) %>%
  step_rename(Year_Remod = Year_Remod_Add) %>%
  step_rename(ThirdSsn_Porch = Three_season_porch) %>%
  step_ratio(Bedroom_AbvGr, denom = denom_vars(Gr_Liv_Area)) %>%
  step_mutate(
    Age_House = Year_Sold - Year_Remod,
    TotalSF   = Gr_Liv_Area + Total_Bsmt_SF,
    AvgRoomSF   = Gr_Liv_Area / TotRms_AbvGrd,
    Pool = if_else(Pool_Area > 0, 1, 0),
    Exter_Cond = forcats::fct_collapse(Exter_Cond, Good = c("Typical", "Good", "Excelle
  step_relevel(Exter_Cond, ref_level = "Good") %>%
  step_normalize(all_predictors(), -all_nominal()) %>%
  step_dummy(all_nominal()) %>%
  step_interact(~ Second_Flr_SF:First_Flr_SF) %>%
  step_interact(~ matches("Bsmt_Cond"):TotRms_AbvGrd) %>%
  step_rm(
    First_Flr_SF, Second_Flr_SF, Year_Remod,
    Bsmt_Full_Bath, Bsmt_Half_Bath,
    Kitchen_AbvGr, BsmtFin_Type_1_Unf,
    Total_Bsmt_SF, Kitchen_AbvGr, Pool_Area,
    Gr_Liv_Area, Sale_Type_Oth, Sale_Type_VWD
  ) %>%
  prep()
```

Recordemos que la función **recipe()** solo son los pasos a seguir, necesitamos usar la función **prep()** que nos devuelve una receta actualizada con las estimaciones y la función **juice()** que nos devuelve la matriz de diseño.

Una vez que la receta de transformación de datos está lista, procedemos a implementar el pipeline del modelo de interés.

```
svm_model <- svm_rbf(
  mode = "regression",
  cost = tune(),
  rbf_sigma = tune(),
  margin = tune()) %>%
```

```
set_engine("kernlab")

svm_workflow <- workflow() %>%
  add_recipe(receta_casas) %>%
  add_model(svm_model)

svm_parameters_set <- parameters(svm_workflow) %>%
  update(
   rbf_sigma = rbf_sigma(c(-2.5, 2.5)),
   cost = cost(c(0, 15))
   )

set.seed(123)
svm_grid <- svm_parameters_set %>%
  grid_max_entropy(size = 80)

ctrl_grid <- control_grid(save_pred = T, verbose = T)
```

```
library(doParallel)

UseCores <- detectCores() - 1
cluster <- makeCluster(UseCores)
registerDoParallel(cluster)

svm1 <- Sys.time()
svm_tune_result <- tune_grid(
  svm_workflow,
  resamples = ames_folds,
  grid = svm_grid,
  metrics = metric_set(rmse, mae, mape),
  control = ctrl_grid
)
svm2 <- Sys.time(); svm2 - svm1

stopCluster(cluster)

svm_tune_result %>% saveRDS("models/svm_model_reg.rds")
```

Podemos obtener las métricas de cada *fold* con el siguiente código:

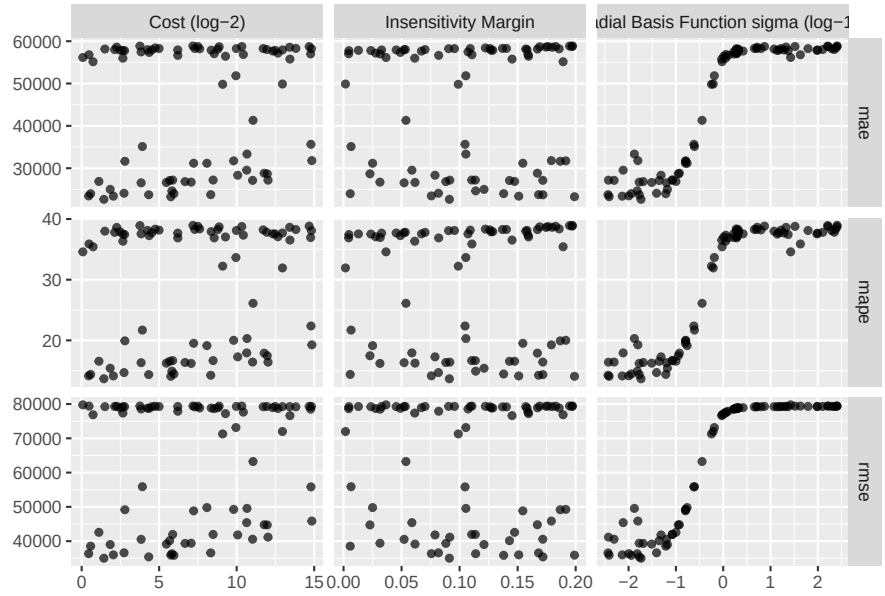
```
svm_tune_result <- readRDS("models/svm_model_reg.rds")

svm_tune_result %>% unnest(.metrics)
```

```
## # A tibble: 2,400 x 11
```

```
##    splits             id      cost rbf_sigma margin .metric .estimator .estimate
##    <list>             <chr>  <dbl>     <dbl>  <dbl> <chr>   <chr>          <dbl>
##  1 <split [1977/220]> Fold~   357.    0.0839 0.110  rmse    standard     43741.
##  2 <split [1977/220]> Fold~   357.    0.0839 0.110  mae     standard     28240.
##  3 <split [1977/220]> Fold~   357.    0.0839 0.110  mape    standard        18.6
##  4 <split [1977/220]> Fold~ 10956.    5.22   0.180  rmse    standard     90721.
##  5 <split [1977/220]> Fold~ 10956.    5.22   0.180  mae     standard     66139.
##  6 <split [1977/220]> Fold~ 10956.    5.22   0.180  mape    standard        42.4
##  7 <split [1977/220]> Fold~  5733.   96.0    0.0128 rmse    standard     91135.
##  8 <split [1977/220]> Fold~  5733.   96.0    0.0128 mae     standard     65465.
##  9 <split [1977/220]> Fold~  5733.   96.0    0.0128 mape    standard        41.4
## 10 <split [1977/220]> Fold~ 26216.    7.45   0.197  rmse    standard     90721.
## # ... with 2,390 more rows, and 3 more variables: .config <chr>, .notes <list>,
## #   .predictions <list>
```

En la siguiente gráfica observamos el error cuadrático medio de las distintas métricas:

```
svm_tune_result %>% autoplot()
```



https://towardsdatascience.com/svm-and-kernel-svm-fed02bef1200

https://towardsdatascience.com/support-vector-machine-explained-8bfef2f17e71

https://medium.com/swlh/the-support-vector-machine-basic-concept-a5106bd3cc5f

https://towardsdatascience.com/unlocking-the-true-power-of-support-vector-regression-847fd123a4a0#:~:text=Support%20Vector%20Regression%20is%20a,the%20maximum%20number%20of%

https://www.mygreatlearning.com/blog/introduction-to-support-vector-machine/

https://www.robots.ox.ac.uk/~az/lectures/ml/lect3.pdf