# Facultad de Ciencias, UNAM
# Reconocimiento de patrones y aprendizaje automatizado
# Tarea 1

Rubí Rojas Tania Michelle

9 de enero de 2021

1. Considera la siguiente expresión para calcular los elementos de un conjunto de punto flotante

$$2(\beta-1)\beta^{l-1}(L-l+1)+1$$

donde $\beta$ es la base, $L$ es el exponente más grande y $l$ el más pequeño. Calcula todos los elementos para los siguientes valores $\beta=2$, $L=2$ y $l=-1$, donde los números tienen 3 cifras significativas.

Construye todos los números de este conjunto F.

Solución:

$$\begin{aligned}
2(\beta-1)\beta^{l-1}(L-l+1)+1 &= 2(2-1)2^{-1-1}(2-(-1)+1)+1 \\
&= 2(3)2^{-2}(3+1)+1 \\
&= 6\left(\frac{1}{4}\right)\ (4)+1 \\
&= \left(\frac{6}{4}\right)5 \\
&= \left(\frac{3}{2}\right)5 \\
&= \frac{15}{2}
\end{aligned}$$

2. Determina el condicionamiento de la siguiente función:

$$f(x)=\sqrt{x-1}-\sqrt{x}$$

3. Implementa los algoritmos de la bisección y de newton en un script. Muestra su funcionamiento con la siguiente función

$$g(x)=x^3-x-1$$

Solución: El algoritmo de `biseccion` fue implementado de la siguiente manera

```
function biseccion(f::Function, a::Number, b::Number,
                   toleranciaEpsilon::AbstractFloat = 1e-10,
                   maxIteracion::Integer=100)
    fa = f(a)
    fa * f(b) <= 0 || error("No existen raíces reales en [a, b].")
    for i = 1:maxIteracion
        p = (a + b)/2 # El punto medio de [a, b].
        fp = f(p)
        # La raíz se encuentra en la mitad derecha (ó izquierda) de [a, b].
        fp * fa > 0 ? a = p : b = p

        if (abs(fp) < toleranciaEpsilon) | ((b - a) < toleranciaEpsilon)
            return p
        end
    end

    error("El número de iteraciones fue excedido.")
end
```

Mostramos su funcionamiento con la función $g(x) = x^3 - x - 1$.

```
# Probamos nuestro algoritmo.
entrada = biseccion(x -> x^3-x-1, 1, 2)
resultado = string(entrada)
println("The resultado es : $resultado")
```

```
The resultado es : 1.324717957235407
```

Por otro lado, el algoritmo de `newton` fue implementado de la siguiente manera

```
function newton(f::Function, dxf::Function, x0::Number, args::Tuple=();
                        toleranciaEpsilon::AbstractFloat=1e-8,
                        maxIteracion::Integer=50, epsilon::AbstractFloat=1e-10)
    for i=1:maxIteracion
        yPrima = dxf(x0, args...)
        if abs(yPrima) < epsilon
            error("La primer derivada es cero.")
        end

        y = f(x0, args...)
        x = x0 - y/yPrima   # Calculamos xi.

        # El procedimiento fue completado satisfactoriamente.
        if abs(x - x0) < toleranciaEpsilon
            return x
        end

        x0 = x  # Redefinimos a x0.
    end

    error("El número de iteraciones fue excedido.")
end
```

Mostramos su funcionamiento con la función $g(x) = x^3 - x - 1$.

```
# Probamos nuestro algoritmo.
entrada = newton(x -> x^3-x-1, x -> 3x^2-1, 1)
resultado = string(entrada)
println("El resultado es: $resultado")
```

```
El resultado es: 1.324717957244746
```

4. Usa los métodos implementados en la pregunta 3 para encontrar $x^*$ que soluciona $g(x^*) = 0$ y $h(x^*) = 0$ en las siguientes funciones

$$g(x) = cos(x) - x$$

en el intervalo $[\frac{1}{2}, 1]$, y

$$h(x) = x^2 - x - 1$$

en el intervalo $[1, 2]$. En este caso, la tolerancia debe ser mínimo de $10^{-8}$.

SOLUCIÓN: Usándo le método de la `biseccion` obtenemos que el valor $x^*$ que soluciona

- $g(x) = cos(x) - x$ es 0.7390.

```
entrada1 = biseccion(x -> cos(x)-x, 0.5, 1)
resultado = string(entrada1)
println("The resultado es : $resultado")
```

```
The resultado es : 0.7390851331874728
```

- $h(x) = x^2 - x - 1$ es 1.6180.

```
entrada2 = biseccion(x -> x^2-x-1, 1, 2)
resultado = string(entrada2)
println("El resultado es: $resultado")
```

```
El resultado es: 1.6180339887505397
```

Comprobamos los resultados obtenidos con el método de `newton`, de tal forma que el valor $x^*$ que soluciona

- $g(x) = cos(x) - x$ es 0.7390

```
entrada3 = newton(x -> cos(x)-x, x -> -sin(x)-1, 1)
resultado = string(entrada3)
println("El resultado es: $resultado")
```

```
El resultado es: 0.7390851332151607
```

- $h(x) = x^2 - x - 1$ es 1.6180

```
entrada4 = newton(x -> x^2-x-1, x -> 2x-1, 2)
resultado = string(entrada4)
println("El resultado es: $resultado")
```

```
El resultado es: 1.618033988749895
```

5. Muestra que el polinomio característico $p(\lambda)$ de una matriz $A \in \mathcal{R}^{2\times 2}$ se puede expresar como

$$p(\lambda) = \lambda^2 - \lambda tr(A) + det(A)$$

donde tr(A) es la traza de la matriz $A$ y det(A) es el determinante de la misma.

*Demostración.* Sea $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in M_{2\times 2}(R)$. Entonces, tenemos que

$$\begin{aligned}
p(\lambda) &= det(A - \lambda I_n) && \text{definición de } p(\lambda) \\
&= det\left(\begin{pmatrix} a & b \\ c & d \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right) && \text{multiplicación escalar de matrices} \\
&= det\left(\begin{pmatrix} a & b \\ c & d \end{pmatrix} - \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix}\right) && \text{sustituyendo valores} \\
&= det\left(\begin{pmatrix} a-\lambda & b \\ c & d-\lambda \end{pmatrix}\right) && \text{resta de matrices} \\
&= (a-\lambda)(d-\lambda) - bc && \text{definición del determinante} \\
&= ad - a\lambda - d\lambda + \lambda^2 - bc && \text{álgebra} \\
&= ad - (a+d)\lambda + \lambda^2 - bc && \text{factorizando} \\
&= \lambda^2 - (a+d)\lambda + (ad - bc) && \text{reacomodando la expresión} \\
&= \lambda^2 - tr(A)\lambda + det(A) && \text{definición de tr(A) y det(A)} \\
&= \lambda^2 - \lambda tr(A) + det(A) && \text{conmutatividad de la multiplicación}
\end{aligned}$$

Por lo tanto, el polinomio característico de la matriz $A$ puede ser expresado como

$$p(\lambda) = \lambda^2 - \lambda tr(A) + det(A)$$

□

6. Utiliza el algoritmo de KNN con el dataset `"trees.csv"`. Este dataset cuenta con tres variables o atributos: el diámetro a la altura del pecho, la altura y el volumen de varios árboles. Utilizándo los notebooks provistos en clase, responde:

   *a*) Define dos variables independientes y una dependiente. Justifica tu elección.

   Solución: Definimos a las variables `"Girth"` y `"Height"` como independientes y a la variable `"Volume"` como dependiente. La elección la realizamos de esta forma porque es más sencillo medir la altura y el diámetro a la altura del pecho, que medir el volúmen de los árboles (estuve leyendo un poco al respecto, y para calcular el volumen sin necesidad de talar el árbol se necesitan algunas técnicas complicadas y muy tardadas). Por lo que, es útil poder predecir el volumen del árbol a partir de la altura y el diámetro dados.

   *b*) Normaliza las variables, ¿para qué hacemos esto?

   Solución: Para que funcionen mejor muchos algoritmos de *Machine Learning* hay que normalizar las variables de entrada al algoritmo. Normalizar significa, en este caso, comprimir o extender los valores de la variable para que estén en un rango definido.

   El escalador **MinMaxScaler** transforma las características escalándolas a un rango dado, por defecto $(0, 1)$, aunque puede ser personalizado. Este tipo de escalado suele denominarse frecuentemente como *normalización* de datos.

```
scaler = MinMaxScaler()

X_train = scaler.fit_transform(x_train)
X_test = scaler.fit_transform(x_test)
```

   *c*) Separa tu dataset en conjunto de entrenamiento y conjunto de prueba. ¿Por qué hacemos esto?

   Solución: Una vez que seleccionamos el mejor modelo que se puede crear con los datos disponibles, se tiene que comprobar su capacidad prediciéndo nuevas observaciones que no se hayan empleado para entrenarlo, de esta forma se verifica si el modelo se puede generalizar. Una estrategia para hacer esto es dividir (de manera aleatoria, o no) los datos en dos grupos, ajustar el modelo con el primer grupo y estimar la precisión de las predicciones con el segundo conjunto.

   El tamaño adecuado para las particiones depende de la cantidad de datos disponibles y la seguridad que se necesite en la estimación del error, pero en general dividir el conjunto en un $80 - 20$ (entrenamiento - prueba) suele dar buenos resultados.

```
# Dividimos nuestro conjunto en entrenamiento y prueba.
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

   En este caso, utilizamos la función `train_test_split` para poder dividir nuestro conjunto de datos en entrenamiento y prueba con los porcentajes mencionamos anteriormente.

   *d*) Encuentra la $k$ óptima para aplicar el algoritmo.

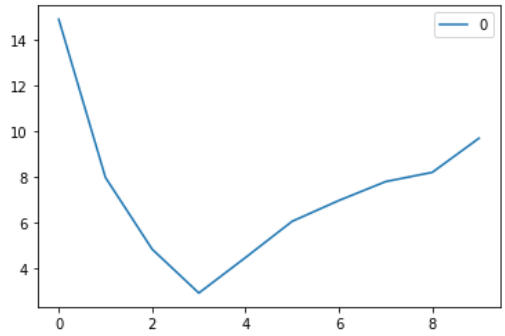
   Solución: Para elegir el valor óptimo de $k$ podemos utilizar la tasa de error obtenida sobre el conjunto de prueba, para así elegir aquella donde encontremos el valor más pequeño. Para lograr esto, aplicaremos el algoritmo probando varios valores de $k$ y en función de ello determinaremos aquella que minimice el error en el conjunto de prueba.

```
rmse_val = [] #to store rmse values for different k
for K in range(10):
    K = K+1
    model = KNeighborsRegressor(n_neighbors = K)

    model.fit(X_train, y_train)  # fit the model
    pred=model.predict(X_test) # make prediction on test set
    error = np.sqrt(mean_squared_error(y_test,pred)) # calculate RMSE (Residuals Mean Squared Error)
    rmse_val.append(error) #store RMSE
    print('RMSE value for k=' , K , 'is:', error)
```

```
RMSE value for k= 1 is: 14.894917637522251
RMSE value for k= 2 is: 7.986394680955854
RMSE value for k= 3 is: 4.85397889479457
RMSE value for k= 4 is: 2.9379596904752017
RMSE value for k= 5 is: 4.488410472698389
RMSE value for k= 6 is: 6.071367880182187
RMSE value for k= 7 is: 6.979192314618611
RMSE value for k= 8 is: 7.804884860045754
RMSE value for k= 9 is: 8.209328156619684
RMSE value for k= 10 is: 9.699529443667432
```

Graficamos los valores obtenidos.




De esta forma, podemos notar que el valor $k = 3$ es el que minimiza el error en el conjunto de prueba, así que aplicaremos el algoritmo con este valor de $k$.

```
# Creamos el modelo con k=3.
knn = KNeighborsRegressor(n_neighbors = 3)
# Entrenamos el modelo.
knn.fit(X_train, y_train)


print('Accuracy of K-NN classifier on training set: {:.2f}'
     .format(knn.score(X_train, y_train)))
print('Accuracy of K-NN classifier on test set: {:.2f}'
     .format(knn.score(X_test, y_test)))
```

```
Accuracy of K-NN classifier on training set: 0.93
Accuracy of K-NN classifier on test set: 0.92
```

```
# Realizamos predicciones sobre el conjunto de prueba.
y_pred = knn.predict(X_test)
```

Comparamos los resultados de las predicciones con sus valores originales.

| | Actual | Predicted |
|---|---|---|
| **2** | 10.2 | 12.066667 |
| **29** | 51.0 | 61.400000 |
| **13** | 21.3 | 21.433333 |
| **10** | 24.2 | 26.966667 |
| **27** | 58.3 | 61.400000 |
| **25** | 55.4 | 61.400000 |
| **22** | 36.3 | 36.266667 |

*e*) Obtén el MSE del modelo calibrado aplicado al conjunto de prueba.
SOLUCIÓN: Para este modelo obtenemos un $MSE$ con valor de 23.56.

```
Mean Absolute Error: 3.47
Mean squared error: 23.56
Root Mean Squared Error: 4.85
Variance score: 0.92
```

7. Utiliza el método de regresión lineal, o en otras palabras, ajusta un modelo lineal a las observaciones del dataset `"trees.csv"`. Utiliza la misma definición de variables independientes y dependientes del ejercicio anterior, así como el mismo conjunto de entrenamiento y de prueba. Responde:

*a*) ¿Cuál es el MSE del modelo lineal que construiste?
SOLUCIÓN: Construimos la regresión múltiple tomando a las variables `Girth, Height` como independientes y a la variable `Volume` como dependiente.
Separamos nuestro conjunto de datos en entrenamiento y prueba, como lo hicimos anteriormente.

```
X_2 = df[['Girth', 'Height']] # Seleccionamos las columnas de Girth y Height
y_2 = df['Volume'] # Seleccionamos la columna de Volume
```

```
X_train2, X_test2, y_train2, y_test2 = train_test_split(X_2, y_2, test_size=0.2, random_state=9)
```

Construimos el modelo.

```
# Creamos el modelo.
modelo_bueno = LinearRegression()
# Entrenamos nuestro modelo.
modelo_bueno.fit(X_train2, y_train2)
```

Visualizamos los coeficientes obtenidos del modelo.

| | Coefficient |
|---|---|
| **Girth** | 4.759537 |
| **Height** | 0.300656 |

Esto significa que cuando la variable `Girth` incrementa en una unidad (pulgada), entonces `Volume` incrementa en 4.75 unidades (pies cúbicos). De manera análoga, cuando `Height` incrementa en una unidad, entonces `Volume` incrementa en 0.3 pies cúbicos. Esto suponiéndo que la otra variable independiente no cambia.
Luego, realizamos predicciones sobre el conjunto de prueba.

```
# Realizamos predicciones usando el conjunto de prueba.
y_pred2 = modelo_bueno.predict(X_test2)
```

Comparamos los resultados de las predicciones con sus valores originales.

| | Actual | Predicted |
|---|---|---|
| **9** | 19.9 | 20.588713 |
| **4** | 18.8 | 20.012880 |
| **23** | 38.3 | 42.532525 |
| **7** | 18.2 | 19.636806 |
| **5** | 19.7 | 21.090146 |
| **3** | 16.4 | 16.355069 |
| **18** | 25.7 | 31.284933 |

Para este modelo obtenemos un $MSE$ con valor de 7.86.

```
Mean Absolute Error: 2.08
Mean squared error: 7.86
Root Mean Squared Error: 2.80
Variance score: 0.84
```

*b*) Comparando el MSE de este modelo con el del modelo anterior, ¿cuál es menor? ¿a qué piensas que se debe?

SOLUCIÓN: El menor valor de $MSE$ obtenido fue de 7.86, el cual corresponde al modelo creado usando regresión lineal.