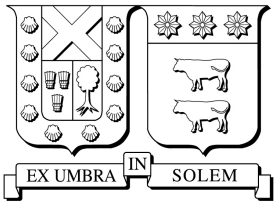



# INF-493. Machine Learning. Tarea 1.
# Métodos Lineales para Regresión

Prof. Ricardo Ñanculef
*jnancu@inf.utfsm.cl*

## Temas

- Manipulación de dataframes en *pandas*.
- Estandarización y normalización de datos.
- Manipulación básica de matrices en *numpy*.
- Regresión lineal multivariada usando *sklearn*.
- Selección de atributos, *from scratch*.
- Regularización (Ridge regression, Lasso) usando *sklearn*.
- Validación cruzada (cross-validation) usando *sklearn*.
- Drift (*transfer learning*).

## Formalidades

- Equipos de trabajo de: 2 personas.*
- Se debe preparar un (breve) Jupyter/IPython notebook que explique la actividad realizada y las conclusiones del trabajo.
- Se debe preparar una presentación de 20 minutos.
- Se debe mantener un respaldo de cualquier tipo de código utilizado, informe y presentación en Github.
- Deadline: Viernes 13 de Octubre.
- Formato de entrega: envío de link Github al correo electrónico del ayudante † incluyendo a todos los profesores en copia y especificando asunto: [TallerXXX-CursoXXX-Semestre-Año].

*La modalidad de trabajo en equipo nos parece importante, porque en base a nuestra experiencia enriquece significativamente la experiencia de aprendizaje. Sin embargo, esperamos que esto no se transforme en una cruda divisi?n de tareas. Cada miembro del equipo debe estar en condiciones de realizar una presentación y discutir sobre cada punto del trabajo realizado.

†francisco.mena.13@sansano.usm.cl

# 1 Regresión Lineal Ordinaria (LSS)

En esta sección trabajaremos con un dataset conocido como *House Sales in King County, USA*, presentado en la plataforma de Kaggle [4], el cual es un gran dataset para evaluar simples modelos de regresión. Los registros contienen distintas características asociadas a las ventas de casas en la localidad King County, entre mayo de 2014 y mayo de 2015, las cuales vienen descritas en el dataset, como la cantidad de habitaciones, cantidad de baños, número de pisos, etc. Donde una de las variables a estudiar corresponde al precio en el cual se vendió la casa.
Para descargar el dataset deben hacerlo mediante el siguiente **link**.

(a) Construya un dataframe con los datos a analizar descargándolos desde la plataforma como se indicó. Explique por qué se realiza la línea 4.

```
1 import pandas as pd
2 import numpy as np
3 df = pd.read_csv("kc_house_data.csv")
4 df.drop(['id','date','zipcode',],axis=1,inplace=True)
5 df.head()
```

(b) Describa brevemente el dataset a utilizar.

```
1 df.shape
2 df.info()
3 df.describe()
```

(b) Normalice los datos antes de trabajar y aplique una transformación adecuada a la variable a predecir. Explique la importancia/conveniencia de realizar estas dos operaciones.

```
1 from sklearn.preprocessing import StandardScaler
2 scaler = StandardScaler()
3 df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
4 df_scaled['price'] = np.log(df['price'])
```

(d) Realice una regresión lineal de mínimos cuadrados básica. Explique la importancia/conveniencia del paso 4 y los argumentos que se deben entregar a la función que implementa la regresión lineal.

```
1  import sklearn.linear_model as lm
2  X = df_scaled.iloc[:,1:]  #use .ix instead, in older pandas version
3  N = X.shape[0]
4  X.insert(X.shape[1], 'intercept', np.ones(N))
5  y = df_scaled['price']
6  #mascara estatica con el 70% de los datos
7  mascara = np.zeros(len(X))
8  limit = int(len(X)*0.7)
9  mascara[:limit] = 1
10 istrain = mascara== 1
11 Xtrain = X[istrain]
12 ytrain = y[istrain]
13 Xtest = X[np.logical_not(istrain)]
14 ytest = y[np.logical_not(istrain)]
15 linreg = lm.LinearRegression(fit_intercept = False)
16 linreg.fit(Xtrain, ytrain)
```

(e) Construya una tabla con los pesos y Z-score correspondientes a cada predictor (variable). ¿Qué variables están más correlacionadas con la respuesta? Si usáramos un nivel de significación del 5%. ¿Qué es lo que observa y cuál puede ser la causa?

```
# do it yourself :-)
```

(f) Proponga un método para corregir lo observado (*Hint:* inspírese en los métodos de *feature engineering* de las siguiente secciones). Verifíquelo mediante los Z-score presentados en la pregunta e).

(g) Estime el error de predicción del modelo usando validación cruzada con un número de *folds* igual a K = 5 y K = 10. Recuerde que para que la estimación sea razonable, en cada configuración (*fold*) deberá reajustar los pesos del modelo. Mida el error real del modelo sobre el conjunto de pruebas, compare y concluya.

```
yhat_test = linreg.predict(Xtest)
mse_test = np.mean(np.power(yhat_test - ytest, 2))
Xm = Xtrain.as_matrix()
ym = ytrain.as_matrix()
from sklearn.model_selection import KFold
kf = KFold(n_splits=10)
mse_cv = 0
for train, val in kf.split(Xm):
    linreg = lm.LinearRegression(fit_intercept = False)
    linreg.fit(Xm[train], ym[train])
    yhat_val = linreg.predict(Xm[val])
    mse_fold = np.mean(np.power(yhat_val - ym[val], 2))
    mse_cv += mse_fold
mse_cv = mse_cv / 10
```

(h) Mida los errores de predicción para cada dato de entrenamiento. Utilizando un "quantile-quantile plot" determine si es razonable la hipótesis de normalidad sobre los residuos del modelo.

## 2 Selección de Atributos

Utilizando el dataframe de la actividad anterior,

(a) Construya una función que implemente Forward Step-wise Selection (FSS). Es decir, partiendo con un modelo sin predictores (variables), agregue un predictor a la vez, re-ajustando el modelo de regresión en cada paso. Para seleccionar localmente una variable, proponga/implemente un criterio distinto al utilizado en el código de ejemplo. Construya un gráfico que muestre el error de entrenamiento y el error de pruebas como función del número de variables en el modelo. Ordene el eje $x$ de menor a mayor.

```
def fss(x, y, names_x, k = 10000):
    p = x.shape[1]-1
    k = min(p, k)
    names_x = np.array(names_x)
    remaining = range(0, p)
    selected = [p]
    current_score = best_new_score = 0.0
    while remaining and len(selected)<=k :
        score_candidates = []
        for candidate in remaining:
            model = lm.LinearRegression(fit_intercept=False)
            indexes = selected + [candidate]
            x_train = x[:,indexes]
            predictions_train = model.fit(x_train, y).predict(x_train)
            residuals_train = predictions_train - y
            mse_candidate = np.mean(np.power(residuals_train, 2))
            score_candidates.append((mse_candidate, candidate))
        score_candidates.sort()
        score_candidates[:] = score_candidates[::-1]
```

```
        best_new_score, best_candidate = score_candidates.pop()
        remaining.remove(best_candidate)
        selected.append(best_candidate)
        print "selected = %s ..."%names_x[best_candidate]
        print "totalvars=%d, mse = %f"%(len(indexes),best_new_score)
    return selected
names_regressors = X.columns[:-1] #without intercept
fss(Xm,ym,names_regressors)
```

# 3 Regularización

Utilizando el dataframe de la actividad anterior,

(a) Ajuste un modelo lineal utilizando "Ridge Regression", es decir, regularizando con la norma $\ell_2$. Utilice valores del parámetro de regularización $\lambda$[‡] en el rango $[10^7, 10^1]$, variando si estima conveniente. Construya un gráfico que muestre los coeficientes obtenidos como función del parámetro de regularización. Describa lo que observa. (WARNING: Note que la línea 3 y el primer argumento en la línea 9 son críticos).

```
from sklearn.linear_model import Ridge
import matplotlib.pylab as plt
X2 = X.drop('intercept', axis=1,inplace=False)
Xtrain = X2[istrain]
ytrain = y[istrain]
names_regressors = X2.columns
alphas_ = np.logspace(7,1,base=10)
coefs = []
model = Ridge(fit_intercept=True,solver='svd')
for a in alphas_:
    model.set_params(alpha=a)
    model.fit(Xtrain, ytrain)
    coefs.append(model.coef_)
ax = plt.gca()
for y_arr, label in zip(np.squeeze(coefs).T, names_regressors):
    plt.plot(alphas_, y_arr, label=label)
plt.legend()
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1]) # reverse axis
plt.title('Regularization Path RIDGE')
plt.axis('tight')
plt.legend(loc=2)
plt.show()
```

(b) Ajuste un modelo lineal utilizando el método "Lasso", es decir, regularizando con la norma $\ell_1$. Utilice valores del parámetro de regularización $\lambda$[§] en el rango $[10^0, 10^{-3}]$. Para obtener el código, modifique las líneas 7 y 9 del ejemplo anterior. Construya un gráfico que muestre los coeficientes obtenidos como función del parámetro de regularización. Describa lo que observa. ¿Es más efectivo Lasso para seleccionar atributos?

```
from sklearn.linear_model import Lasso
alphas_ = np.logspace(0,-3,base=10)
model = Lasso(fit_intercept=True)
```

[‡]Se asume la siguiente formulación: $\min_{\boldsymbol{w}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{w}\|^2 + \lambda\|\boldsymbol{w}\|^2$.
[§]Se asume la siguiente formulación: $\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{w}\|^2 + \lambda\|\boldsymbol{w}\|_{\ell_1}$.

(c) Escogiendo uno de los dos métodos regularizadores anteriores, especificando el porqué, construya un gráfico que muestre el error de entrenamiento y el error de pruebas como función del parámetro de regularización. Discuta lo que observa.

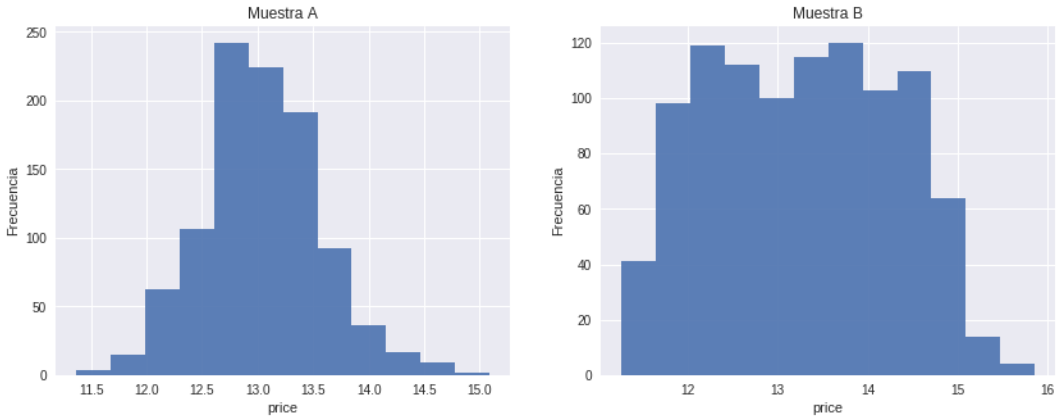
```
Xtest = X2[np.logical_not(istrain)]
ytest = y[np.logical_not(istrain)]
alphas_ = #choose it
coefs = []
model = #choose it
mse_test = []
mse_train = []
for a in alphas_:
    model.set_params(alpha=a)
    model.fit(Xtrain, ytrain)
    yhat_train = model.predict(Xtrain)
    yhat_test = model.predict(Xtest)
    mse_train.append(np.mean(np.power(yhat_train - ytrain, 2)))
    mse_test.append(np.mean(np.power(yhat_test - ytest, 2)))
ax = plt.gca()
ax.plot(alphas_,mse_train,label='train error ridge')
ax.plot(alphas_,mse_test,label='test error ridge')
plt.legend(loc=1)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])
plt.show()
```

(e) Estime el valor del parámetro de regularización en **alguno** de los modelos anteriores haciendo uso de la técnica validación cruzada.

```
def MSE(y,yhat): return np.mean(np.power(y-yhat,2))
Xm = Xtrain.as_matrix()
ym = ytrain.as_matrix()
from sklearn.model_selection import KFold
kf = KFold(n_splits=10)
best_cv_mse = float("inf")
model = #choose it
alphas_ = #alphas to evaluate
for a in alphas_:
    model.set_params(alpha=a)
    mse_list_k10 = [MSE(model.fit(Xm[train], ym[train]).predict(Xm[val]), ym[val]) \
            for train, val in kf.split(Xm)]
    if np.mean(mse_list_k10) < best_cv_mse:
        best_cv_mse = np.mean(mse_list_k10)
        best_alpha = a
        print "BEST PARAMETER=%f, MSE(CV)=%f"%(best_alpha,best_cv_mse)
```

# 4 Drift

En esta sección se presentarán dos muestras del dataframe utilizado en la actividades anteriores, donde cada una de estas tiene una propiedad distinta ya que son muestreadas en función del valor a predecir (logaritmo del precio de la casa). Por una parte se tiene una pequeña muestra A, la cual es extraída directamente de los datos con los que se trabaja (manteniendo la distribución de esta) y la muestra B, es generada con el propósito de que en cada intervalo del rango de valores haya la misma cantidad de datos aproximadamente (simulando una distribución uniforme). El objetivo es familiarizarse con el concepto de *Transfer Learning*[2]




En el siguiente código se generan las dos muestras con las que se trabajará.

```
df_A = df_scaled.sample(1000,random_state=11)

frames = []
valor = df_scaled.price
length = 0.3
for z in np.arange(int(np.min(valor)),int(np.max(valor))+1,length):
    #un maximo de 100 datos por intervalo
    aux = df_scaled[(df_scaled.price >= z) & (df_scaled.price < z+length)].head(100)
    frames.append(aux)
df_B = pd.concat(frames).sample(1000,random_state=11) #crea el dataframe
```

(a) Cree el conjunto de entrenamiento y otro de validación para trabajar cada muestra mediante la técnica de *hold out validation*.

```
X_A = df_A.iloc[:,1:].values
y_A = df_A.price
X_B = df_B.iloc[:,1:].values
y_B = df_B.price
from sklearn.model_selection import train_test_split
Xtrain_A,Xval_A,ytrain_A,yval_A = train_test_split(X_A, y_A, test_size=0.3, random_state=42)
Xtrain_B,Xval_B,ytrain_B,yval_B = train_test_split(X_B, y_B, test_size=0.3, random_state=42)
```

(b) Evalúe los dos modelo de regresión lineal que se generan al entrenar con cada muestra. Mida el error de cada modelo sobre ambos conjuntos de validación (A y B). Explique lo que observa.

(c) Si tuviera que elegir uno de los dos modelos anteriores para trabajar con data futura, ¿Cuál eligiría y por qué?

# 5 Detectar enfermedades cardícas

En el área de la salud, diagnosticar a una persona de una enfermedad de forma rápida y correcta puede llegar a salvarle la vida. Los encargados de realizar estos diagnósticos, son médicos que, observando exámenes y ciertos indicadores, pueden concluir qué enfermedad presenta el paciente. Si él medico se llegase a equivocar, aparte de que el paciente pueda perder la vida, el medico podría ser demandado por negligencia arriesgando años de cárcel o pagar sumas de dinero considerable, es por estas razones que es importante no cometer errores.
Pongámonos en el contexto de que usted es contratado para generar un modelo que prediga si es que un paciente presenta una enfermedad cardiaca a partir de ciertos indicadores, tales como la edad, sexo, presión

sanguínea, nivel de glicemia, etc. Los datos para trabajar junto a su documentación pueden ser descargados ejecutando los siguientes comandos en un terminal (sistemas UNIX)

```
wget https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/heart/heart.dat
wget https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/heart/heart.doc
```

Como ayuda se le indica que la variable de máximo ritmo cardíaco alcanzado (*maximum heart rate achieved*) es un buen indicador de detección de enfermedades cardíacas. Por lo que el objetivo es predecir el comportamiento de esta variable en función de las otras, y con esto detectar qué tan distante es el valor real al valor predecido para así luego detectar los posibles *outliers* (enfermos), que en sí corresponden a pacientes que tienen un comportamiento anormal al resto.

(a) Lea el archivo de datos, cárguelos en un dataframe o matriz, luego divida el dataframe en dos, un dataframe de entrenamiento (70% Datos) y un dataframe de prueba (30% Datos).

```
headers = ['age','sex','chest_pain','blood_p','serum','blood_s','electro','max_heart', \
                        'angina','oldpeak','slope','vessel','thal','normal']
df = pd.read_csv(dataset, header=None, names=headers, sep=' ')
#create your matrix
```

(b) Realice una regresión lineal y defina usted una frontera de decisión (umbral) para poder determinar si es que estamos en presencia o no de una enfermedad cardíaca. Mida su desempeño con ambos conjuntos de datos.

```
from sklearn.metrics import accuracy_score
print "Score: "%(accuracy_score(y_outlier,y_predict_outlier)
```

# References


[1] Hastie, T.; Tibshirani, R., Friedman, J. (2009), The Elements of Statistical Learning, Second Edition. Springer New York Inc.

[2] Torrey, L., & Shavlik, J. (2009). Transfer learning. Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques, 1, 242.

[3] http://ruder.io/tag/transfer-learning/

[4] https://www.kaggle.com