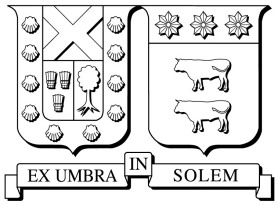

Análisis Inteligente de Datos - Tarea II

Prof. Ricardo Ñanculef
jnancu@inf.utfsm.cl

Instrucciones

- El objetivo de estas tareas es que pongan en práctica los métodos discutidos en clases. El énfasis debe ser puesto por lo tanto en la comprensión del problema abordado, la implementación computacional de los algoritmos correspondientes, las decisiones prácticas que haya tenido que tomar para completar el análisis y la interpretación de los resultados.
- Lo ideal es que hagan esta tarea en parejas. Cada miembro del equipo debe estar en condiciones de exponer cada punto del trabajo realizado.
- Se recomienda el uso de Python como lenguaje de programación, junto a todo el ecosistema de librerías que hemos mencionado en clases, incluyendo: pandas, numpy, sklearn, statsmodels y seaborn.
- La realización de la tarea consiste en preparar un breve informe y presentación (slides) del trabajo realizado. Por favor, incluya en estos productos sólo aquello esencial para dar a entender lo realizado. Los programas utilizados contendrán todos los detalles. Éstos deben ser entregados y mantenidos en un repositorio público como github.
- La fecha entrega de todo el material entregable corresponde al día Jueves 22 de Junio.
- El formato de entrega será electrónico vía email a jnancu@inf.utfsm.cl.

1 Regresión Lineal Ordinaria (LSS)

En esta sección trabajaremos con un pequeño dataset conocido como *prostate-cancer*, utilizado con frecuencia para realizar pruebas preliminares con métodos de regresión. Los datos vienen de un famoso estudio publicado en 1989 por Tom Stamey, profesor de Urología de la Universidad de Stanford, sobre la eventual correlación entre el nivel de antígeno prostático específico (PSA) medido en un paciente, y una serie de otras mediciones clínicas obtenidas después de practicar al paciente una prostatectomía radical, i.e., extirpación total de la próstata y tejidos circundantes. Una de las variables estudiadas corresponde al volumen de cáncer prostático detectado en el paciente (log transformed).

(a) Construya un dataframe con los datos a analizar descargando los datos desde la URL mantenida por los autores de [1]. Explique qué hacen las líneas 5 a 9.

```
1 import pandas as pd
2 import numpy as np
3 url = 'http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/prostate.data'
4 df = pd.read_csv(url, sep='\t', header=0)
5 df = df.drop('Unnamed: 0', axis=1)
```

```
istrain_str = df['train']
istrain = np.asarray([True if s == 'T' else False for s in istrain_str])
istest = np.logical_not(istrain)
df = df.drop('train', axis=1)
```

(b) Describa brevemente el dataset utilizar.

```
df.shape
df.info()
df.describe()
```

(c) Normalice los datos antes de trabajar. Explique la importancia/conveniencia de realizar esta operación.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
df_scaled['lpsa'] = df['lpsa']
```

(d) Realice una regresión lineal de mínimos cuadrados básica. Explique la importancia/conveniencia del paso 3 y los argumentos que se deben entregar a la función que implementa la regresión lineal.

```
import sklearn.linear_model as lm
X = df_scaled.ix[:,:-1]
N = X.shape[0]
X.insert(X.shape[1], 'intercept', np.ones(N))
y = df_scaled['lpsa']
Xtrain = X[istrain]
ytrain = y[istrain]
Xtest = X[np.logical_not(istrain)]
ytest = y[np.logical_not(istrain)]
linreg = lm.LinearRegression(fit_intercept = False)
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%, ¿Para qué variables no existe suficiente evidencia que demuestre la relación con la respuesta?

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

(f) 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 debe ajustar los pesos del modelo de nuevo, cada vez que trabaja sobre un determinado "fold". 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))
from sklearn import cross_validation
Xm = Xtrain.as_matrix()
ym = ytrain.as_matrix()
k_fold = cross_validation.KFold(len(Xm),10)
mse_cv = 0
for k, (train, val) in enumerate(k_fold):
        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
```

(j) 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 = 0.0
        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 = ["Lcavol", "Lweight", "Age", "Lbph", "Svi", "Lcp", "Gleason", "Pgg45"]
fss(Xm,ym,names_regressors)
```

(b) Construya una función que implemente Backward Step-wise Selection (BSS). Es decir, partiendo con un modelo completo (todas las variables), elimine un predictor a la vez, re-ajustando el modelo de regresión en cada paso. 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 mayor a menor.

```
# do it yourself (note that it is quite easy to modify FSS to obtain BSS)
```

3 Regularización

Utilizando el dataframe de la actividad anterior,

(a) Ajuste un modelo lineal utilizando "Ridge Regression", es decir, regularizando con la norma ℓ_2. Utilice valores del parámetro de regularización λ[‡] en el rango $[10^4, 10^{-1}]$. 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).

```
1  from sklearn.linear_model import Ridge
2  import matplotlib.pylab as plt
3  X = X.drop('intercept', axis=1)
4  Xtrain = X[istrain]
5  ytrain = y[istrain]
6  names_regressors = ["Lcavol", "Lweight", "Age", "Lbph", "Svi", "Lcp", "Gleason", "Pgg45"]
7  alphas_ = np.logspace(4,-1,base=10)
8  coefs = []
9  model = Ridge(fit_intercept=True,solver='svd')
10 for a in alphas_:
11         model.set_params(alpha=a)
12         model.fit(Xtrain, ytrain)
13         coefs.append(model.coef_)
14 ax = plt.gca()
15 for y_arr, label in zip(np.squeeze(coefs).T, names_regressors):
16         print alphas_.shape
17         print y_arr.shape
18         plt.plot(alphas_, y_arr, label=label)
19 plt.legend()
20 ax.set_xscale('log')
21 ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
22 plt.xlabel('alpha')
23 plt.ylabel('weights')
24 plt.title('Regularization Path RIDGE')
25 plt.axis('tight')
26 plt.legend(loc=2)
27 plt.show()
```

(b) Ajuste un modelo lineal utilizando el método "Lasso", es decir, regularizando con la norma ℓ_1. Utilice valores del parámetro de regularización λ[§] en el rango $[10^1, 10^{-2}]$. 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?

```
1  from sklearn.linear_model import Lasso
2  alphas_ = np.logspace(1,-2,base=10)
3  clf = Lasso(fit_intercept=True)
```

(c) Utilizando "Ridge Regression", construya un gráfico que muestre el error de entrenamiento y el error de pruebas como función del parámetro de regularización. Discuta lo que observa.

```
1  Xtest = X[np.logical_not(istrain)]
2  ytest = y[np.logical_not(istrain)]
3  alphas_ = np.logspace(2,-2,base=10)
4  coefs = []
5  model = Ridge(fit_intercept=True)
6  mse_test = []
7  mse_train = []
8  for a in alphas_:
```

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

```
        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=2)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])
plt.show()
```

(d) Utilizando "Lasso", construya un gráfico que muestre el error de entrenamiento y el error de pruebas como función del parámetro de regularización. Discuta lo que observa.

```
alphas_ = np.logspace(0.5,2,base=10)
```

(e) Estime el valor del parámetro de regularización en los métodos anteriores usando validación cruzada.

```
def MSE(y,yhat): return np.mean(np.power(y-yhat,2))
Xm = Xtrain.as_matrix()
ym = ytrain.as_matrix()
k_fold = cross_validation.KFold(len(Xm),10)
best_cv_mse = float("inf")
model = Lasso(fit_intercept=True)
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 k_fold]
        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 Predicción de Utilidades de Películas

El problema a resolver en esta sección consiste en predecir el volumen de utilidades (en dólares) obtenidas por el estreno (al público, en USA) de una película. Específicamente consideraremos dos posibles respuestas: el volumen total de utilidades (total revenue) obtenido durante el fin de semana del estreno y el volumen de utilidades por lugar de proyección (per screen revenue). Los datos a utilizar fueron recolectados en un estudio publicado recientemente por M. Joshi y colegas de la universidad de Carnegie Mellon [2], y corresponden a 1718 películas realizadas entre 2005 y 2009. Cada película, se representa utilizando diversos tipos de atributos

1. Texto: A partir de las críticas publicadas para cada película (en diversos sitios y antes del estreno), se construyen características que corresponden a la frecuencia de palabras, parejas de palabras y tríos de palabras obtenidas de un vocabulario.

2. Metadata: (1) Variable binaria que indica si el lugar de origen de la película es USA, (2) logaritmo del presupuesto, (3) número de puntos de proyección, (4) género (acción, drama, comedia, etc), (5) Calificación de la MPAA (mayores de catorce, todo espectador, etc), (6) Variable binaria que indica si el estreno se produjo durante un feriado/vacaciones y (7) Número de actores con OSCAR.

Los datos estarán disponibles en el link [3]. Para facilitar el trabajo, los datos han sido ya preparados en formato matricial. Concretamente usted dispondrá de tres parejas de archivos (X e y): un conjunto de datos

de entrenamiento (*train.x.nm* y *train.y.dat*), un conjunto de datos de validación para evitar tener que hacer validación cruzada (*dev.x.nm* y *dev.y.dat*) y un conjunto de datos de pruebas (*test.x.nm* y *test.y.dat*) que, naturalmente, no puede utilizar para construir el modelo. Se incluirán además, dos versiones de los datos. Una de ellas consiste en remover la variable que indica la presencia de actores con OSCAR del conjunto de atributos.

El archivo de respuestas (y) contiene un dato por fila en el orden correspondiente a las x. El archivo con los atributos (x) está codificado en formato *sparse* de *matrix market* [4] como sigue. La cabecera del archivo pueden aparecer $0, 1$ o más comentarios (filas que inician con %). La linea siguiente indica el número de filas, columnas y entradas no nulas de la matriz. Las líneas que siguen tienen la estructura (i, j, X_{ij}), es decir indican la fila y columna de la matriz que contiene el tercer elemento. Por ejemplo:

```
%%MatrixMarket matrix coordinate real general
317 145256 658516
1 9 14.0
1 12 1.0
```

(a) Lea los archivos de datos y cárguelos en dos dataframe o matrices X, y. En el caso de X es extremadamente importante que mantenga el formato disperso (sparse) (¿porqué?). Si trabaja con matrices use matrices dispersas del tipo *csr_matrix* o *csc_matrix*. Si prefiere operar sobre un dataframe, puede utilizar los (recientemente introducidos) dataframe dispersos de pandas: *SparseDataFrame*, aunque todavía no se tiene una operabilidad completa.

```
import pandas as pd
import numpy as np
from scipy.sparse import csr_matrix
from scipy.io import mmread
X = csr_matrix(mmread('test.x.nm'))
y = np.loadtxt('test.y.dat')
```

(b) Construya un modelo lineal que obtenga un coeficiente de determinación (sobre el conjunto de pruebas) de al menos 0.75. A partir de un modelo lineal de *sklearn*, el coeficiente de determinación se obtiene fácilmente

```
import sklearn.linear_model as lm
model = lm.LinearRegression(fit_intercept = False)
print "R2=%f"%model.score(X, y)
```

References

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

[2] Joshi, M., Das, D., Gimpel, K., Smith, N. A. (2010). Movie reviews and revenues: An experiment in text regression. In the 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (pp. 293-296). Association for Computational Linguistics.

[3] https://www.dropbox.com/sh/8r1wrblyfokwuq0/AABUEvgcuMxyZht2-KYyBptUa?dl=0

[4] http://math.nist.gov/MatrixMarket/formats.html

LaTeX