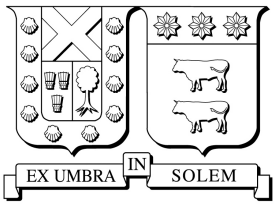


INF-493. Machine Learning. Tarea 3.
Métodos No-Lineales.

Prof. Ricardo Ñanculef & Carlos Valle
jnancu@inf.utfsm.cl

Temas

- Manipulación de dataframes en *pandas*.
- Pre-procesamiento de datos y extracción de características.
- SVMs no-lineales en *sklearn*.
- Árboles de clasificación en *sklearn*.
- Redes Neuronales feed-forward en *keras*.
- Ensamblados de máquinas de aprendizaje en *sklearn*.
- Selección de hiper-parámetros estructurales vía validación cruzada.

Formalidades

- Equipos de trabajo de: 2 personas.[1].
- 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: 2 semanas desde publicación.
- 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].

Paquetes/Librerías

Como es usual utilizaremos *numpy*, *scipy*, *matplotlib* y *sklearn*.

Para esta tarea necesitará además instalar *keras*, una librería para prototipado rápido de modelos basados en redes neuronales usando pyhton, muy similar en espíritu a *sklearn*. La librería puede usar *Tensor Flow* o *Theano* como backend, siendo éstas las librerías más populares para desarrollar nuevos modelos de redes neuronales o implementar eficientemente modelos conocidos con fines aplicativos. Para detalles sobre la instalación puede revisar [1] o escribir un email a su ayudante.

[1]Cada miembro del equipo debe estar en condiciones de realizar una presentación y discutir sobre cada punto del trabajo realizado.

1 El Viejo XOR: Métodos No-lineales para Problemas No-lineales

El objetivo de esta sección es experimentar con algunos modelos no-lineales sobre un problema de juguete, pero bastante famoso en la historia del aprendizaje automático, que denominaremos *XOR*. Se trata de un problema de clasificación a todas luces *linealmente inseparable*, en el sentido que, si denotamos por $\mathbf{x} \in \mathbb{R}^2$ un patrón de entrada y por $y \in \{0, 1\}$ su correspondiente etiqueta, no existen $\mathbf{w} \in \mathbb{R}^2, b \in \mathbb{R}$ tal que $y(\mathbf{w}^T\mathbf{x} + b) \geq \rho > 0$. El problema nos permite hace un recorrido rápido por las grandes ideas en la búsqueda de la *no-linealidad*.

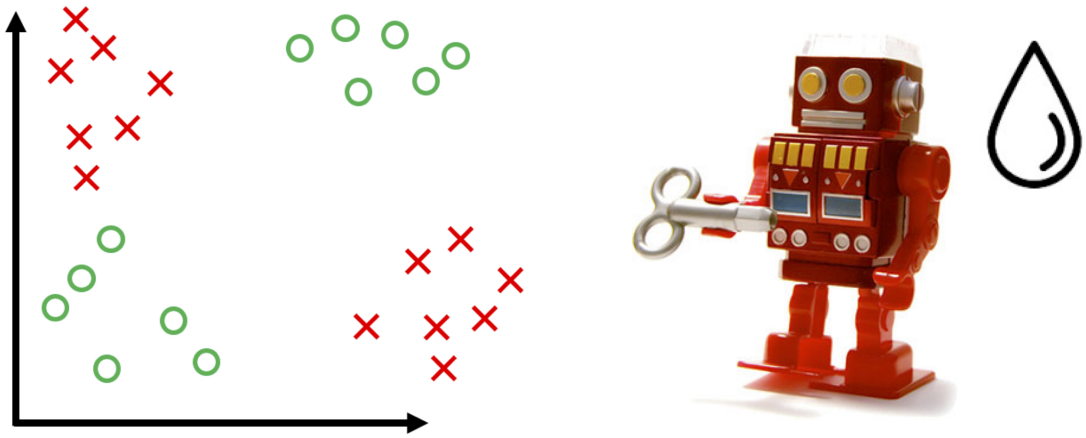


Figure 1: Distribución deseada para la actividad 1. Los 2 colores representan 2 clases distintas.

(a) Escriba una función que genere (aleatoriamente) n datos etiquetados de la forma $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, $\mathbf{x}_i \in \mathbb{R}^2$, $y_i \in \{0, 1\}$, con una distribución de probabilidad que refleje la configuración linealmente inseparable que muestra la Fig.1[2]. Utilice esta función para crear 1000 datos de entrenamiento y 1000 datos de pruebas. Para medir la tendencia de los modelos a *sobre-ajuste*, agregue un 10% de ruido al dataset de entrenamiento, generando $\mathbf{x}$'s cercanos a la frontera e invirtiendo la etiqueta correcta de este subconjunto. Genere in gráfico que muestre sus datos de entrenamiento y pruebas, identificando cada clase con un color diferente. ¿Porqué tiene sentido denominar este problema *XOR*?

```
import numpy as np

def do_XOR(n=1000,noisy_n=100,svm=True):
        rng = np.random.RandomState(0)
        X_train = rng.uniform(low=-1.0,high=1.0,size=(n,2))
        Y_train = np.logical_xor(X_train[:,0] > 0, X_train[:,1] > 0)
        Y_train = 2*Y_train-1 if svm else Y_train
        X_noisy= rng.uniform(low=-0.8,high=0.2,size=(noisy_n,2))
        Y_noisy = -1*np.logical_xor(X_noisy[:,0] > 0, X_noisy[:,1] > 0) + 1
        Y_noisy = 2*Y_noisy-1 if svm else Y_noisy
        X_train = np.vstack((X_train, X_noisy))
        Y_train = np.hstack((Y_train, Y_noisy))
        X_test = rng.uniform(low=-1.0,high=1.0,size=(n,2))
        Y_test = np.logical_xor(X_test[:,0] > 0, X_test[:,1] > 0)
        Y_test = 2*Y_test - 1 if svm else Y_test
        return X_train,Y_train,X_test,Y_test

X_train,Y_train,X_test,Y_test = do_XOR()
```

[2]Puede generar datos aleatorios distribuidos uniformemente en $[-1, +1] \times [-1, +1]$ para luego etiquetar aquellos ubicados en el primer y tercer cuadrante como 0 y aquellos ubicados en el primer y tercer cuadrante como 1.

(b) Demuestre experimentalmente que una SVM lineal no puede resolver satisfactoriamente el problema anterior [3]. Sea convincente: por ejemplo, intente modificar los parámetros de la máquina de aprendizaje, reportando métricas que permitan evaluar el desempeño del modelo en el problema. Escriba también una función, denominada *plot_classifier* que represente gráficamente la solución encontrada por la máquina de aprendizaje, representando los datos de entrenamiento y pruebas (asociando un color a la clase que les corresponde), además de la frontera de clasificación. Describa y explique lo que observa, reportando gráficos de la solución sólo para algunos casos representativos.

```
import matplotlib.pyplot as plt

def plot_classifier(clf,X_train,Y_train,X_test,Y_test,model_type):
    f, axis = plt.subplots(1, 1, sharex='col', sharey='row',figsize=(20, 20))
    axis.scatter(X_train[:,0],X_train[:,1],s=5,c=Y_train,zorder=10,cmap='gray')
    axis.scatter(X_test[:,0],X_test[:,1],s=15,c=Y_test,zorder=10,cmap='gray')
    XX, YY = np.mgrid[-1:1:200j, -1:1:200j]
    if model_type == 'svm':
        Z = clf.decision_function(np.c_[XX.ravel(), YY.ravel()])
    elif model_type == 'tree':
        Z = clf.predict_proba(np.c_[XX.ravel(), YY.ravel()])[:,0]
    elif model_type == 'ann':
        Z = clf.predict(np.c_[XX.ravel(), YY.ravel()])
    else: raise ValueError('model type not supported')
    Z = Z.reshape(XX.shape)
    Zplot = Z > 0 if model_type == 'svm' else Z > 0.5
    axis.pcolormesh(XX, YY, Zplot ,cmap='YlGn')
    axis.contour(XX, YY, Z, alpha=1, colors=['k', 'k', 'k'],
                            linestyles=['--', '-', '--'],levels=[-1, 0, 1])
    plt.show()
```

(c) Demuestre experimentalmente que una SVM con kernel no-lineal puede resolver satisfactoriamente el problema. Para esta actividad, use los hiper-parámetros que se entregan como referencia en el código de ejemplo (muestre resultados tanto para el kernel *Polynomial* como para el kernel *RBF*). Cambie el valor del parámetro C eligiendo κ en el intervalo $[-2, 4]$ (con saltos de 1 unidad) y asignando al parámetro el valor $C = 2^{\kappa}$. Describa y explique lo que observa, graficando el error de entrenamiento y pruebas como función de C. Utilice la función *plot_classifier*, diseñada en el ítem anterior, para construir gráficos de la solución con distintos valores de C.

```
from sklearn.svm import SVC
clf = SVC(C=100, kernel='rbf')
clf = SVC(C=10, kernel='poly',degree=2, coef0=1)
clf.fit(X_train, Y_train)
print "Test Accuracy = %f"%clf.score(X_test,Y_test)
plot_classifier(clf,X_train,Y_train,X_test,Y_test,'svm')
```

(d) Demuestre experimentalmente que una red neuronal artificial correspondiente a 1 sola neurona (i.e. sin capas escondidas) no puede resolver satisfactoriamente el problema. Puede utilizar la función de activación y el método de entrenamiento que prefiera. Sea convincente: por ejemplo, intente modificar los parámetros de la máquina de aprendizaje, reportando métricas que permitan evaluar el desempeño del modelo en el problema con cada cambio efectuado. Adapte también la función *plot_classifier* para que represente gráficamente la solución encontrada por la red neuronal. Describa y explique lo que observa, reportando gráficos de la solución sólo para algunos casos representativos.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
n_h=1
model = Sequential()
```

[3] Defina satisfactorio como al menos 50% mejor que una predicción aleatoria de 1 de las clases.

```
model.add(Dense(1, input_dim=X_train.shape[1], init='uniform', activation='relu'))
model.add(Dense(n_h, init='uniform', activation='sigmoid'))
model.compile(optimizer=SGD(lr=1), loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, Y_train, nb_epoch=50, batch_size=100, verbose=1)
scores = model.evaluate(X_test, Y_test)
test_acc = scores[1]
```

(e) Demuestre experimentalmente que una red neuronal artificial con 1 capa escondida puede resolver satisfactoriamente el problema obtenido en (a). Puede utilizar la arquitectura y el método de entrenamiento que prefiera, pero en esta actividad puede optar tranquilamente por usar los hiper-parámetros que se entregan como referencia en el código de ejemplo. Cambie el número de neuronas N_h en la red entre 2 y 32 en potencias de 2, graficando el error de entrenamiento y pruebas como función de N_h. Describa y explique lo que observa. Utilice la función *plot_classifier*, diseñada anteriormente, para construir gráficos de la solución en algunos casos representativos.

```
...
n_h=8
model = Sequential()
model.add(Dense(1, input_dim=X_train.shape[1], init='uniform', activation='relu'))
model.add(Dense(n_h, init='uniform', activation='sigmoid'))
```

(f) Demuestre experimentalmente que stump (árbol de clasificación de 1 nivel) no puede resolver satisfactoriamente el problema anterior. Puede utilizar el criterio y la función de partición que prefiera. Sea convincente: por ejemplo, intente modificar los parámetros de la máquina, reportando métricas que permitan evaluar el desempeño del modelo en el problema con cada cambio efectuado. Adapte también la función *plot_classifier* para que represente gráficamente la solución encontrada por el árbol. Describa y explique lo que observa, reportando gráficos de la solución sólo para algunos casos representativos.

```
from sklearn.tree import DecisionTreeClassifier as Tree
clf=Tree(criterion='gini',splitter='best',random_state=0,max_depth=1)
clf.fit(X_train,Y_train)
acc_test = clf.score(X_test,Y_test)
print "Test Accuracy = %f"%acc_test
print clf.tree_.max_depth
plot_classifier(clf,X_train,Y_train,X_test,Y_test,'tree')
```

(g) Demuestre experimentalmente que un árbol de clasificación de múltiples niveles puede resolver satisfactoriamente el problema estudiado. Puede utilizar el criterio y la función de partición que prefiera, pero puede optar tranquilamente por usar los hiper-parámetros que se entregan como referencia en el código de ejemplo. Cambie el número de niveles admitidos en el árbol N_t entre 2 y 20, graficando el error de entrenamiento y pruebas como función de N_t. Describa y explique lo que observa. Utilice la función *plot_classifier*, diseñada anteriormente, para construir gráficos de la solución en algunos casos representativos.

```
...
n_t=8
clf=Tree(criterion='gini',splitter='best',random_state=0,max_depth=n_t)
clf.fit(X_train,Y_train)
```

2 Bike Sharing: Predicción de Demanda Horaria

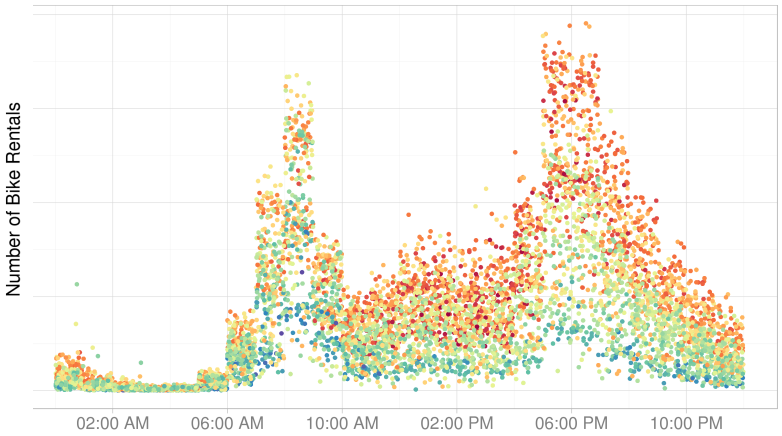
En esta sección simularemos nuestra participación en el desafío *Bike Sharing Demand* de *Kaggle* [2]. El objetivo es predecir la demanda de bicicletas sobre la red *Capital Bikeshare* de la ciudad de Washington, D.C., en función de la hora del día y otras variables descritas en la tabla 1. En principio, y como muestra la figura, la función es altamente no lineal y no determinista como función de de la hora del día. Su objetivo será entrenar un modelo para obtener un puntaje correspondiente al top-100 del "leaderboard" final, es decir superior o igual a 0.37748. La función utilizada para evaluar este concurso Kaggle se proporciona en la siguiente ecuación:

$$E_{\text{bikes}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n} \sum_i \left(\ln(y_i + 1) - \ln(\hat{y}_i + 1)\right)^2 \ , \tag{1}$$

donde $\boldsymbol{y}, \hat{\boldsymbol{y}} \in \mathbb{R}^n$ denotan los vectores de observaciones y predicciones respectivamente.

Como el dataset de pruebas original no está disponible se fabricará uno, correspondiente al 20% de los datos de entrenamiento. Además, se pondrá a su disposición un subconjunto independiente de datos con propósitos de validación. Usted podrá descargar los archivos correspondientes al subconjunto de entrenamiento y pruebas a utilizar ejecutando los siguientes comandos:

```
wget http://octopus.inf.utfsm.cl/~ricky/bike_sharing_train.csv
wget http://octopus.inf.utfsm.cl/~ricky/bike_sharing_val.csv
wget http://octopus.inf.utfsm.cl/~ricky/bike_sharing_test.csv
```



Atributo	Descripción
datetime	hourly date + timestamp
season	1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday	whether the day is considered a holiday
workingday	whether the day is neither a weekend nor holiday
weather	1: Clear, Few clouds, Partly cloudy, Partly cloudy
	2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
	3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
	4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp	temperature in Celsius
atemp	"feels like" temperature in Celsius
humidity	relative humidity
windspeed	wind speed
casual	number of non-registered user rentals initiated
registered	number of registered user rentals initiated
count	number of total rentals

Table 1: Atributos para el Problema 2 (*Bike Sharing*).

(a) Cargue los datos de entrenamiento y pruebas como dataframes de *pandas*. Describa las variables involucradas en el problema, explorando el tipo de datos de que se trata, el número de valores distintos y, si corresponde, un gráfico (e.g. un histograma) que resuma su comportamiento. Su primera operación de pre-procesamiento de datos será obtener la hora del día desde el campo fecha (que en este momento es de tipo string), creando una nueva columna denominada *hour* y de tipo *int*. Para hacer esta operación se concatenarán los dataframes de entrenamiento y pruebas y luego se volverán a separar manteniendo la separación original.

```
import pandas as pd
import numpy as np
dftrain = pd.read_csv('bike_sharing_train.csv')
dfval = pd.read_csv('bike_sharing_val.csv')
dftest = pd.read_csv('bike_sharing_test.csv')
ntrain = len(dftrain)
nval =  len(dftrain) +  len(dfval)
df = pd.concat([dftrain,dfval,dftest])
print('\nSummary - dataframe completo:\n')
print(df.describe())
df['hour'] = pd.to_datetime(df['datetime']).apply(lambda x: x.strftime('%H'))
df['hour'] = pd.to_numeric(df['hour'])
```

(b) Entrene un árbol de regresión para resolver el problema usando parámetros por defecto. Con este fin, construya una matriz $\boldsymbol{X}_{\text{train}}$ de forma $n_{\text{train}} \times d_1$ que contenga los datos de entrenamiento en sus filas, seleccionando las columnas que desee/pueda utilizar para el entrenamiento. Implemente además, la función de evaluación que hemos definido anteriormente para este problema. Evalúe el árbol de regresión ajustado a los datos de entrenamiento sobre el conjunto de entrenamiento y pruebas. Construya un gráfico que compare las predicciones con los valores reales. En este punto usted debiese tener un modelo con puntaje del orden de 0.59, lo que lo dejaría más o menos en la posición 2140 de la competencia.

```
from sklearn.tree import DecisionTreeRegressor as Tree
import matplotlib.pyplot as plt
def eval_bikemodel(y_predict,y_true):
        diff = np.log(y_predict+1.0) - np.log(y_true+1.0)
        return np.sqrt(np.sum(np.square(diff))/len(y_predict))
Xdf=df.ix[:,['season','holiday','workingday','weather','temp','atemp',
                'humidity','windspeed','hour']]
Ydf=df.ix[:,'count']
X_train = Xdf[0:ntrain].values
X_val = Xdf[ntrain:nval].values
X_test = Xdf[nval:].values
Y_train = Ydf[0:ntrain].values
Y_val = Ydf[ntrain:nval].values
Y_test = Ydf[nval:].values

model = Tree(random_state=0)
model.fit(X_train,Y_train)
score_test = model.score(X_test,Y_test)
print "SCORE TEST=%f"%score_test

Y_pred_train = model.predict(X_train)
Y_pred_val =  model.predict(X_val)
Y_pred_test = model.predict(X_test)
kagg_train = eval_bikemodel(Y_pred_train,Y_train)
kagg_val = eval_bikemodel(Y_pred_val,Y_val)
kagg_test = eval_bikemodel(Y_pred_test,Y_test)
print "KAGG EVAL TRAIN =%f"%kagg_train
print "KAGG EVAL TEST =%f"%kagg_test
```

```
plt.plot(Y_test,Y_pred_test,'.')
plt.show()
```

(c) Mejore el árbol de regresión definido en el punto anterior haciendo modificaciones a los hiper-parámetros del modelo. Por ejemplo, como estos modelos tienden a sobre-ajustar, podría intentar limitar la profundidad del árbol (porqué esto debiese ayudar?). Naturalmente, está absolutamente prohibido tomar este tipo de decisiones en función del resultado de pruebas. Debe realizar estas elecciones evaluando sobre el conjunto de validación. Si no desea utilizarlo, y prefiere implementar validación cruzada u otra técnica automática, tiene la ventaja de poder usar el conjunto de validación como parte del entrenamiento. Con estas modificaciones debiese poder mejorar su ranking en unas 300 posiciones.

```
model = Tree(random_state=0,max_depth=20)
Y_pred_val =  model.predict(X_val)
kagg_val = eval_bikemodel(Y_pred_val,Y_val)
print "KAGG EVAL VAL =%f"%kagg_val
```

(d) Mejore el árbol de regresión definido en el punto anterior haciendo modificaciones sobre la representación utilizada para aprender desde los datos. Por ejemplo, los histogramas que construyó en el punto (a) así como la forma especial de la función de evaluación, sugieren una cierta transformación de la variable respuesta. Podría intentar también normalizando los datos o normalizando la respuesta. Otra opción es intentar rescatar algo más acerca de la fecha (anteriormente sólo se extrajo la hora), como por ejemplo el año o el día de la semana ('lunes','martes', etc) que corresponde. Sea creativo, este paso le debiese reportar un salto de calidad muy significativo. Una observación importante es que si hace una transformación a la variable respuesta (por ejemplo raíz cuadrada), debe invertir esta transformación antes de evaluar el desempeño con *eval_bikemodel* (por ejemplo, elevar al cuadrado si tomó raíz cuadrada). Con modificaciones de este tipo, podría mejorar su ranking en unas 1000 posiciones, entrando ya al top-1000 con un score del orden de 0.45.

```
df['cday'] = pd.to_datetime(df['datetime']).dt.dayofweek#0:lunes,6:domingo
df['cday'] = pd.to_numeric(df['cday'])
Xdf=df.ix[:,['season','holiday','workingday','weather','temp','atemp',
                'humidity','windspeed','hour','cday']]
```

(e) Entrene una SVM no lineal para resolver el problema midiendo el efecto de las distintas representaciones que haya descubierto hasta este punto. Un detalle importante es que antes de entrenar la SVM sería aconsejable hacer dos tipos de pre-procesamiento adicional de los datos: (i) codificar las variables categóricas en un modo apropiado - por ejemplo como vector binario con un 1 en la posición del valor adoptado-, (ii) escalar los atributos de modo que queden centrados y con rangos comparables. Usando parámetros por defecto para la SVM debiese obtener un score del orden de 0.344, quedando definitivamente en el top-10 de la competencia.

```
#load dataframes as before ...
df = pd.concat([dftrain,dfval,dftest])
df['hour'] = pd.to_datetime(df['datetime']).apply(lambda x: x.strftime('%H'))
df['cday'] = pd.to_datetime(df['datetime']).dt.dayofweek
df['hour'] = pd.to_numeric(df['hour'])
df['cday'] = pd.to_numeric(df['cday'])
Xdf=df.ix[:,['season','holiday','workingday','weather','temp','atemp',
                'humidity','windspeed','hour','cday']]
#PASO IMPORTANTE MAS ABAJO ...
Xdf = pd.get_dummies(Xdf,columns=['season', 'weather','hour','cday'])
Ydf=df.ix[:,'count']

from sklearn.preprocessing import StandardScaler
scalerX = StandardScaler()
X_train = scalerX.fit_transform(X_train)
X_val = scalerX.fit_transform(X_val)
X_test = scalerX.transform(X_test)
```

```
18
19 from sklearn.svm import SVR
20 model = SVR()
21 model.fit(X_train,Y_train)
22 Y_pred_train = model.predict(X_train)
23 Y_pred_val =  model.predict(X_val)
24 Y_pred_test = model.predict(X_test)
```

(f) Mejore la SVM definida en el punto anterior haciendo modificaciones a los hiper-parámetros de la máquina (C, ϵ o la misma función de kernel). Naturalmente, está absolutamente prohibido tomar este tipo de decisiones de diseño mirando el resultado de pruebas. Debe realizar estas elecciones evaluando sobre el conjunto de validación. Si no desea utilizarlo, y prefiere implementar validación cruzada u otra técnica automática, tiene la ventaja de poder usar el conjunto de validación como parte del entrenamiento.

```
1 model = SVR(C=1,epsilon=0.01)
2 kagg_train = eval_bikemodel(Y_pred_train,Y_train)
3 kagg_val = eval_bikemodel(Y_pred_val,Y_val)
4 print "KAGG EVAL TRAIN =%f"%kagg_train
5 print "KAGG EVAL VAL =%f"%kagg_val
```

(g) Evalúe el efecto de utilizar el dataset de validación para entrenamiento y seleccionar los parámetros estructurales del árbol de clasificación y la SVM usando validación cruzada. El código de ejemplo para esto ha sido proporcionado en las tareas 1 y 2, pero se adjunta de nuevo a continuación

```
1 from sklearn import cross_validation
2 k_fold = cross_validation.KFold(len(X_train),10)
3 score_cv = 0
4 for k, (train, val) in enumerate(k_fold):
5         model = #define your model, e.g. model = SVR(C=candidate)
6         model.fit(X_train[train], Y_train[train])
7         Ypred_val = model.predict(X_train[val])
8         Ytrue_val = Y_train[val]
9         score_fold = eval_bikemodel(Ypred_val,Ytrue_val)
10        score_cv += score_fold
11 score_cv = score_cv / 10
```

(h) Evalúe el efecto de utilizar un ensamblado de 2 máquinas de aprendizaje para predecir la demanda total de bicicletas. Un modelo se especializaría en la predicción de la demanda de bicicletas de parte de usuarios registrados y otra en la predicción de la demanda de usuarios casuales. Hay razones claras para pensar que los patrones son distintos.

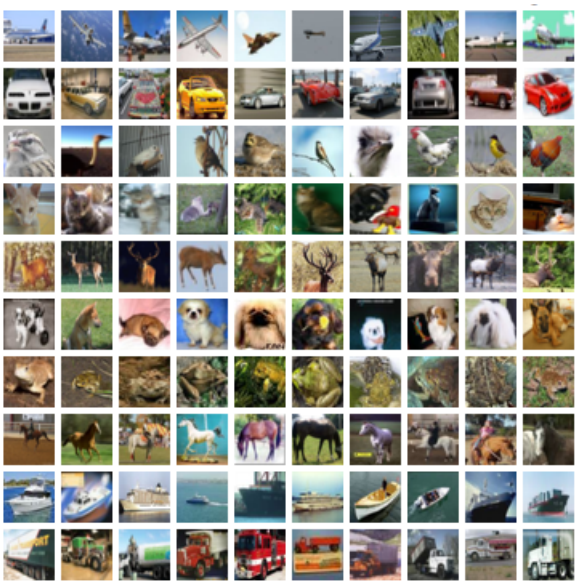
```
1 Ydf=df.ix[:,'count'] #demanda total
2 Ydf=df.ix[:,'registered'] #demanda registrada
3 Ydf=df.ix[:,'casual'] #demanda casual
```

(i) Evalúe el efecto de utilizar un algoritmo genérico para ensamblar máquinas de aprendizaje para predecir la demanda total de bicicletas. Puede experimentar con una sola técnica (e.g. *Random Forest*), pero por favor estudie y discuta la evolución del rendimiento (de entrenamiento y pruebas) a medida que aumenta el número de máquinas.

```
1 from sklearn.ensemble import RandomForestRegressor
2 model = RandomForestRegressor(n_estimators=10,max_depth=max_depth,random_state=0)
```

3 Reconocimiento de Imágenes en CIFAR10

En esta sección trabajaremos con un dataset muy utilizado para experimentar con reconocimiento de imágenes: CIFAR10. Se trata de un conjunto de 60.000 imágenes RGB de 32×32 pixeles que contiene 10 clases de objetos (6000 ejemplos por clase). La versión utilizada se atribuye a [5] y viene separada en 50000 ejemplos de entrenamiento y 10000 casos de prueba. El conjunto de pruebas fue obtenido seleccionando 1000 imágenes aleatorias de cada clase. Los datos restantes han sido ordenados aleatoriamente y están organizados en 5 bloques de entrenamiento (batches). Las clases son mutuamente excluyentes y corresponden a las siguientes categorías: gato, perro, rana, caballo, pájaro, ciervo, avión, automóvil, camión y barco.



Los datos asociados a esta actividad podrán ser obtenidos utilizando los siguientes comandos en la línea de comandos (sistemas UNIX)

```
wget http://octopus.inf.utfsm.cl/~ricky/data.tar.gz
tar -xzvf data.tar.gz
rm data.tar.gz
```

En la carpeta generada encontrarán 5 archivos denominados 'data_batch_1', 'data_batch_2', 'data_batch_3', 'data_batch_4', 'data_batch_5' y 'test_batch' correspondientes a los 5 bloques de entrenamiento y al conjunto de pruebas respectivamente. Los archivos corresponden a diccionarios serializados de python y pueden ser "extraídos" utilizando la siguiente función:

```
def unpickle(file):
    import cPickle
    fo = open(file, 'rb')
    dict = cPickle.load(fo)
    fo.close()
    return dict
```

Una vez extraído, cada diccionario contendrá 2 elementos importantes: *data* y *labels*. El primer elemento (*data*) es un matriz de 10000×3072 (numpy array). Cada fila de esa matriz corresponde a una imagen RGB: los primeros 1024 valores vienen del canal R, los siguientes 1024 del canal G, y los últimos 1024 del canal B. Para cada canal, las imágenes han sido vectorizadas por filas, de modo que los primeros 32 valores del canal R corresponden a la primera fila de la imagen. Por otro lado, el elemento (*labels*) del diccionario contiene una lista de 1000 valores enteros entre 0 y 9 que identifican las clases antes enumeradas.

```
label_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog',\
                        'frog', 'horse', 'ship', 'truck']
```

(a) Construya una función que cargue todos los bloques de entrenamiento y pruebas del problema CIFAR generando como salida: (i) dos matrices X_{tr}, Y_{tr}, correspondientes a las imágenes y etiquetas de entrenamiento, (ii) dos matrices X_t, Y_t, correspondientes a las imágenes y etiquetas de pruebas, y finalmente (iii) dos matrices X_v, Y_v, correspondientes a imágenes y etiquetas que se usarán como conjunto de validación, es decir para tomar decisiones de diseño acerca del modelo. Este último conjunto debe ser extraído desde el conjunto de entrenamiento original y no debe superar las 10000 imágenes.

```
from scipy.misc import imread
import cPickle as pickle
import numpy as np
import os

def load_CIFAR_one(filename):
  with open(filename, 'rb') as f:
    datadict = pickle.load(f)
    X = datadict['data']
    Y = datadict['labels']
    return X, np.array(Y)

def load_CIFAR10(PATH):
  xs = [], ys = []
  for b in range(1,6):
    f = os.path.join(PATH, 'data_batch_%d' % (b, ))
    X, Y = load_CIFAR_one(f)
    xs.append(X)
    ys.append(Y)
  Xtr = np.concatenate(xs)
  Ytr = np.concatenate(ys)
  del X, Y
  Xte, Yte = load_CIFAR_batch(os.path.join(PATH, 'test_batch'))
  return Xtr, Ytr, Xte, Yte
#you need to add Xval
load_CIFAR10('.')
```

(b) Construya una función que escale apropiadamente las imágenes antes de trabajar. Experimente sólo escalando los datos de acuerdo a la intensidad máxima de pixel (i.e., dividiendo por 255) y luego centrando y escalándolos como en actividades anteriores.

(c) Diseñe, entrene y evalúe una red neuronal para el problema CIFAR a partir de la representación original de las imágenes (píxeles RGB). Experimente con distintas arquitecturas y métodos de entrenamiento, midiendo el error de clasificación sobre el conjunto de validación. En base a esta última medida de desempeño, decida qué modelo, de entre todos los evaluados, evaluará finalmente en el conjunto de test. Reporte y discuta los resultados obtenidos. Se espera que logre obtener un error de pruebas menor o igual a 0.5.

```
from keras.utils.np_utils import to_categorical
from sklearn.preprocessing import StandardScaler

def scaler_function(Xtr,Xt,scale=True):
    scaler = StandardScaler(with_std=scale).fit(Xtr)
    Xtr_scaled = scaler.transform(Xtr)
    Xt_scaled = scaler.transform(Xt)
    return Xtr_scaled, Xt_scaled

Xtr,Xte = scaler_function(Xtr,Xte)
Ytr = to_categorical(Ytr)
Yte = to_categorical(Yte)

```

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

model = Sequential()
model.add(Dense(100, input_dim=Xtr.shape[1], init='uniform', activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(10, init='uniform', activation='softmax'))
model.compile(optimizer=SGD(lr=0.05), loss='binary_crossentropy', metrics=['accuracy'])
model.fit(Xtr, Ytr, nb_epoch=50, batch_size=32, verbose=1, validation_data=(Xte,Yte))
scores = model.evaluate(Xte, Yte)
test_acc = scores[1]
```

(d) Repita la actividad anterior, pero mejorando los atributos utilizados para representar las imágenes. Para esta parte, se distribuirá junto a esta tarea una función denominada *extract_features.py* que extraerá 2 tipos de representaciones sobre una imagen y conjunto de imágenes: (i) histogramas de tono [4], (ii) descriptores HOG [3]. Reporte y discuta los resultados obtenidos utilizando las distintas representaciones por separado o todas simultáneamente. La función *extract_features.py* estará definida en un script denominado *top_level_features.py* y puede ser importada y utilizada como se muestra a continuación.

```
from top_level_features import hog_features
from top_level_features import color_histogram_hsv
from top_level_features import extract_features
Xtr, Ytr, Xte, Yte = load_CIFAR10("datasets/")
features = extract_features(Xtr,[hog_features]) #extrae hog features
features = extract_features(Xtr,[color_histogram_hsv]) #extrae histogramas de color
features = extract_features(Xtr,[hog_features, color_histogram_hsv]) #extrae todo
print Xtr.shape
print features.shape
```

(e) Entrene una SVM no lineal sobre los pixeles originales y sobre los atributos de alto nivel utilizados para representar las imágenes en el item anterior. Puede utilizar el conjunto de validación para seleccionar hiper-parámetros, como el nivel de regularización aplicado y/o la función de kernel a utilizar.

(f) Entrene una árbol de clasificación sobre los pixeles originales y sobre los atributos de alto nivel utilizados para representar las imágenes en el item anterior. Puede utilizar el conjunto de validación para seleccionar hiper-parámetros, como la profundidad máxima del árbol.

References

[1] Keras: Deep Learning library for Theano and TensorFlow. https://keras.io/

[2] https://www.kaggle.com/c/bike-sharing-demand

[3] Dalal, N., Triggs, B. (2005, June). Histograms of oriented gradients for human detection. In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) (Vol. 1, pp. 886-893). IEEE.

[4] Forsyth, D. A., Ponce, J. (2002). Computer vision: a modern approach. Prentice Hall Professional Technical Reference.

[5] Krizhevsky, A., Hinton, G. (2009). Learning multiple layers of features from tiny images.