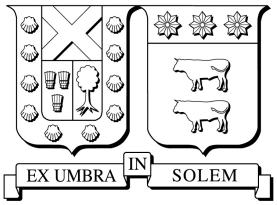


INF-393/INF-578 Machine Learning. Tarea 3.

Métodos No-Lineales.

Prof. Ricardo Ñanculef -*jnancu@inf.utfsm.cl*
Prof. Carlos Valle - *cvalle@inf.utfsm.cl*
Pablo Ibarra S - *pablo.ibarras@alumnos.usm.cl*
Francisco Mena - *francisco.mena.13@sansano.usm.cl*

Temas

- Pre-procesamiento de datos y extracción de características.
- Árboles de clasificación en *sklearn*.
- Redes Neuronales feed-forward en *keras*.
- Métodos de *Kernel*
- 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.
- En la clase Recuperativa del Martes 19 de Diciembre, el grupo (todos los integrantes) debe venir preparado para presentar cada punto de la tarea. Puede utilizar Jupiter, diapositivas, o cualquier material de apoyo.
- Se debe mantener un respaldo de cualquier tipo de código utilizado, informe y presentación en Github.
- Deadline: Martes 19 Diciembre.
- Formato de entrega: envío de link Github al correo electrónico del ayudante [2] incluyendo a todos los profesores en copia y especificando asunto: [TallerXXX-CursoXXX-Semestre-Año]

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

[2] francisco.mena.13@sansano.usm.cl

Paquetes/Librerías

Como es usual utilizaremos *numpy*, *scipy*, *matplotlib* y *sklearn*. Además de éstas se necesitará instalar *keras*, una librería en python para prototipado rápido de modelos basados en redes neuronale, muy similar en espíritu a *sklearn*. La librería puede usar *TensorFlow* o *Theano* como backend, siendo éstas las librerías más populares para desarrollar nuevos modelos de redes neuronales o implementar eficientemente modelos conocidos con fines prácticos. Para detalles sobre la instalación puede revisar [1] o escribir un email a su ayudante.

1. Small Circle inside Large Circle

El objetivo de esta sección es experimentar con algunos modelos no-lineales sobre un problema de juguete generado para visualizar algoritmos de *clustering*. 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 $y \in \{-1, 1\}$ 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 hacer un recorrido rápido por las grandes ideas en la búsqueda de la *no-linealidad*.

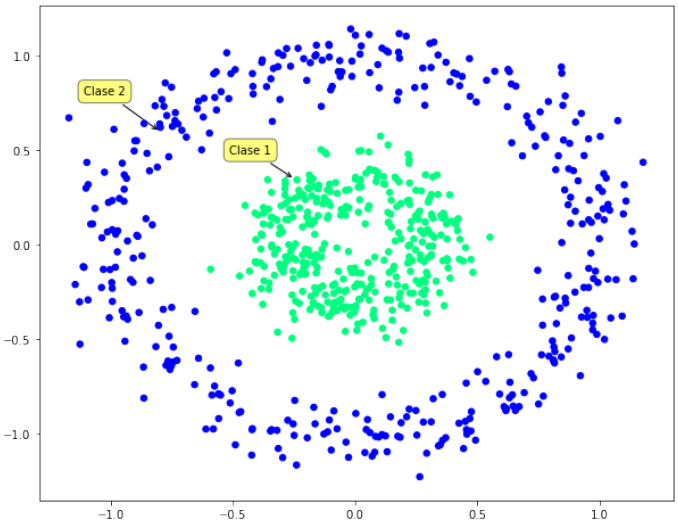


Figura 1: Distribución deseada para la actividad 1. Los 2 colores representan 2 clases distintas.

(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[3]. 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 5 % de ruido al dataset, generando $\mathbf{x}$'s cercanos a la frontera. Genere un gráfico que muestre datos de entrenamiento y pruebas, identificando cada clase con un color diferente (como lo muestra la Fig. 1).

```
import numpy as np
from sklearn.utils import check_random_state
from sklearn.model_selection import train_test_split
def do_circles(n=2000,noisy_n=0.05):
    generator = check_random_state(10)
    linspace = np.linspace(0, 2 * np.pi, n // 2 + 1)[:-1]
    outer_circ_x = np.cos(linspace)
    outer_circ_y = np.sin(linspace)
```

[3]Puede generar datos aleatorios distribuidos de manera circular para luego etiquetar aquellos ubicados en el círculo interior como 1 y en el círculo exterior 0.

```
    inner_circ_x = outer_circ_x * .3
    inner_circ_y = outer_circ_y * .3
    X = np.vstack((np.append(outer_circ_x, inner_circ_x),
                   np.append(outer_circ_y, inner_circ_y))).T
    y = np.hstack([np.zeros(n // 2, dtype=np.intp),
                   np.ones(n // 2, dtype=np.intp)])
    X += generator.normal(scale=noisy_n, size=X.shape)

    X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                        test_size=0.5, random_state=42)
    return X_train,y_train,X_test,y_test
```

Para lo que sigue de la actividad utilice la siguiente función para graficar las fronteras de clasificación en base a la probabilidad, definida por un algoritmo, de un ejemplo a pertenecer a una clase en particular.

```
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=(12, 8))
    axis.scatter(X_train[:,0],X_train[:,1],s=30,c=Y_train,zorder=10,cmap='cool')
    axis.scatter(X_test[:,0],X_test[:,1],s=20,c=Y_test,zorder=10,cmap='Greys')
    XX, YY = np.mgrid[-2:2:200j, -2:2:200j]
    if 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 >= 0.5
    axis.pcolormesh(XX, YY, Zplot ,cmap='YlGn')
    axis.contour(XX, YY, Z, alpha=1, colors=["k", "k", "k"], linestyles=["--", "-", "--"],
    levels=[-2, 0, 2])
    plt.show()
```

(b) 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()
model.add(Dense(1, input_dim=X_train.shape[1], kernel_initializer='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, epochs=50, batch_size=100, verbose=1)
scores = model.evaluate(X_test, Y_test)
test_acc = scores[1]
```

(c) 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=32
model = Sequential()
model.add(Dense(n_h, input_dim=X_train.shape[1], kernel_initializer='uniform', activation='relu'))
model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))
```

(d) 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')
```

(e) 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)
```

(f) Como ya se demostró experimentalmente que este problema es linealmente inseperable, ahora se pide experimentar otra alternativa. Para ello deberá realizar una proyección de los datos a un nuevo espacio dimensional (*manifold*) en el cual se reconozcan sus patrones no lineales, para poder trabajarlos con fronteras lineales. Utilice la técnica de PCA con la ayuda de un *Kernel Gaussiano* ([2]) para extraer sus vectores con dimensión *infinita* de mayor varianza.

```
kpca = KernelPCA(n_components=2,kernel="rbf", gamma=5)
kpca = kpca.fit(X_train)
Xkpca_train = kpca.transform(X_train)
Xkpca_test = kpca.transform(X_test)
```

(g) Ajuste un algoritmo de aprendizaje con fronteras lineal para los datos proyectados en este nuevo espacio que captura sus componentes no lineales, muestre graficamente que el problema ahora puede ser resulto con estos métodos. Reporte métricas para evaluar el desempeño, comente y concluya.

2. Bike Sharing: Predicción de Demanda Horaria

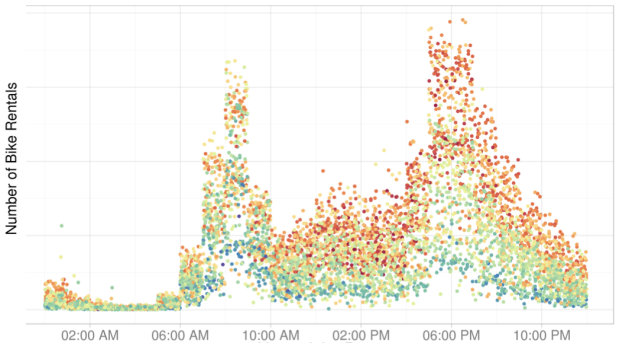
En esta sección simularemos nuestra participación en el desafío *Bike Sharing Demand* de *Kaggle* [3]. El objetivo es predecir la demanda de bicicletas sobre la red *Capital Bikeshare* de la ciudad de Washington, D.C., en función de la hora del día y otras variables descritas en la tabla 1. En principio, y como muestra la figura, la función es altamente no lineal y no determinista como función de 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

Tabla 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 dejará 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 (¿Por qué 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)
model.fit(X_train,Y_train)
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)

from sklearn.svm import SVR
model = SVR()
model.fit(X_train,Y_train)
Y_pred_train = model.predict(X_train)
Y_pred_val =  model.predict(X_val)
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.

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

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=10)
mse_cv = 0
for train, val in kf.split(Xm):
        model = #define your model
        model.fit(Xm[train], ym[train])
        yhat_val = model.predict(Xm[val])
    ytrue_val = ym[val]
    score_fold = eval_bikemodel(yhat_val,ytrue_val)
        mse_cv += score_fold
mse_cv = mse_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 especializará 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.

```
Ydf=df.ix[:,'count'] #demanda total
Ydf=df.ix[:,'registered'] #demanda registrada
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*), discuta la evolución a medida que aumenta el número de máquinas.

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

3. Calidad de un vino

Dentro de las muchas variedades de vino existentes, algunas gustan más que otras, esto es debido al gusto de una persona en particular o bien a la gran cantidad de químicos y procesos que se aplican a la producción de vino. Para el área de negocios, el estimar cuál es la calidad de un vino en base a la apreciación del público es una tarea bastante difícil.
Para esta actividad se trabajará con dos datasets asociados a las variantes tinto y blanco del vino portugués "Vinho Verde"[4]. Debido a temas privados solo se cuenta con las característcas fisioquímicas asociadas a un vino en particular, los cuales corresponden a 11 atributos numéricos descritos en el siguiente link.
Este problema puede ser abordado como clasificación de 11 clases o de regresión, ya que el atributo a estimar, *quality*, es un valor entero entre 0 y 10.

(a) Carge los dos dataset en un único dataframe de pandas, además de agregar una columna indicando si es vino tinto o blanco. Describa el dataset a trabajar.

```
import pandas as pd
df_red = pd.read_csv("winequality-red.csv",sep=";")
df_white = pd.read_csv("winequality-white.csv",sep=";")
df = pd.concat([df_red,df_white], axis=0)
#genere atributo 'tipo'
```

(b) Aborde este problema como si fuera de clasificación binaria para predecir si un vino es de buena calidad o no, es decir, utilice las distintas características fisioquímicas presentes en los datos para estimar esta etiqueta. Para esto cree las matrices de entrenamiento y de pruebas, además de la etiqueta para ambos conjuntos, considerando como *quality* mayor a 5 un vino de buena calidad. El conjunto de pruebas (25 %) será utilizado únicamente para verificar la calidad de los algoritmos a entrenar.

```
df['good_quality'] = [1 if q>5 else 0 for q in df.quality]
#train and test split over df
```

(c) Entrene un solo Árbol de Clasificación de múltiples niveles para resolver el problema. Puede variar los hiper-parámetros que prefiera, recuerde que las decisiones no pueden ser basadas mirando el conjunto de pruebas. Debido al desbalanceo que se produce en las dos clases mida la métrica *F1-score* [5] sobre el conjunto de entrenamiento y de pruebas.

(d) Entrene un ensamblador de árboles de múltiples niveles, mediante la técnica de *Random Forest*. Varíe la cantidad de árboles de decisión utilizados en el ensamblado (*n_estimators*), realice un gráfico resumen del *F1-score* de entrenamiento y de pruebas en función de este hiper-parámetro.

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=, max_depth=,n_jobs=-1)
from sklearn.metrics import f1_score
f1_score(y_true, y_pred)
```

(e) Entrene un ensamblador de árboles de múltiples niveles, mediante la técnica de *AdaBoost*. Varíe la cantidad de árboles de decisión utilizados en el ensamblado (*n_estimators*), realice un gráfico resumen del *F1-score* de entrenamiento y de pruebas en función de este hiper-parámetro. Compare y analice con la técnica utilizada en d).

```
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(base_estimator=Tree(max_depth=), n_estimators=)
```

(f) Entrene alguna otra máquina de aprendizaje, elegida por usted, para resolver este problema. Elija los hiper-parámetros que estime convenientes intentando aumentar el *F1-score* obtenido por los algoritmos anteriores. Compare y analice estas 4 maneras de resolver el problema definido en b).

(g) Defina un criterio para estimar la importancia de los distintos atributos en el ensamblado de *Random Forest*, implementelo sobre alguno de los ensambladores entrenados en d), haga un ranking de importancia de atributos ¿Es posible implementar este criterio sobre una técnica de *boost* como lo es *AdaBoost*?

4. Reconocimiento de Imágenes Sign Gestures

MNIST es un *dataset* muy popular de dígitos escrito a mano que a servido para probar distintos algoritmos de *Machine Learning* relacionados con *Computer Vision*. Buscando nuevos desafíos, investigadores generaron un dataset que podría usarse eventualmente en aplicaciones reales, Sign Gestures, consta de imagenes del lenguaje de señas, estas tienen una resolución de 28x28 pixeles representados en una escala de grises 0-255. La versión utilizada se atribuye a [8] y viene separada en 27455 ejemplos de entrenamiento y 7172 casos de pruebas. Las clases son mutualmente excluyentes y corresponden a las letras del alfabeto (ver imagen).

(a) Construya una función que cargue todos los datos de entrenamiento y pruebas del problema 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 7000 imágenes.

```
import pandas as pd
from sklearn.model_selection import train_test_split
def load_data():
        train = pd.read_csv('sign_mnist_train.csv')
        test = pd.read_csv('sign_mnist_test.csv')
        y_tr = train['label']
        x_tr = train.iloc[:,1:]
        y_t = test['label']
        x_t = test.iloc[:,1:]
    #you need to add Xval: x_v,y_v
        return(x_tr,x_v,x_t,y_tr,y_v,y_t)
x_tr, x_v, x_t, y_tr, y_v , y_t= load_data()
```

(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 partir de la representación original de las imágenes. Experimente con distintas arquitecturas, pre-procesamientos 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, medirá 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 0.2.

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

model = Sequential()
model.add(Dense(30, input_dim=x_tr.shape[1], init='uniform', activation='relu'))
model.add(Dense(30, init='uniform', activation='relu'))
model.add(Dense(25, init='uniform', activation='softmax'))
model.compile(optimizer=SGD(lr=0.05), loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_tr.values, to_categorical(y_tr), nb_epoch=100, batch_size=128, verbose=1,
validation_data=(x_v.values,to_categorical(y_v)))
```

(d) Para la mejor red entrenada anteriormente construya la matriz de confusión de las distintas clases, para asi visualizar cuáles son las clases más difíciles de clasificar y con cuáles se confunden. Comente.

(e) Entrene una SVM no lineal sobre los pixeles con y sin pre-procesamiento. 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 con y sin pre-procesamiento. Puede utilizar el conjunto de validación para seleccionar hiper-parámetros, como la profundidad máxima del árbol.

Referencias

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

[2] Bernhard Schoelkopf, Alexander J. Smola, and Klaus-Robert Mueller. 1999. Kernel principal component analysis. In Advances in kernel methods, MIT Press, Cambridge, MA, USA 327-352.

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

[4] http://archive.ics.uci.edu/ml/datasets/Wine+Quality

[5] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

[6] 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.

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

[8] https://www.kaggle.com/datamunge/sign-language-mnist