Clase11 IMA539

Alejandro Ferreira Vergara

April 24, 2023

1 Conexión al Servidor de Cómputo Khipu

- **Alejandro Ferreira Vergara**
- **Departamento de Ingeniería Matemática y Minor de Análisis de Datos**

- **Primer Semestre 2023**

En esta clase aprenderemos sobre cómo conectarse al Servidor Khipu. Además, aprenderemos a instalar y utilizar correctamente `conda` en él.

1.1 Sobre Khipu

- **Sistema Operativo:** Linux 22.04.
- **CPU:** 24 cores, 48 hilos.
- **Memoria RAM:** 256 GB.
- **Almacenamiento:** 18 TB SSD.
- **GPU:** 4 *Nvidia RTX A4000.*

1.2 Datos de conexión

```
IP SERVIDOR : 200.13.6.14
PUERTO      : 10022
```

Protocolo SSH

SSH o Secure Shell, es un protocolo de administración remota que permite a los usuarios controlar y modificar sus servidores remotos a través de Internet mediante de un mecanismo de autenticación.

Proporciona un mecanismo para autenticar un usuario remoto, transferir entradas desde el cliente al host y retransmitir la salida de vuelta al cliente. Utiliza técnicas de criptografía para garantizar que todas las comunicaciones hacia y desde el servidor remoto sucedan de manera encriptada.

Para conectarse al Servidor Khipu, se debe abrir una terminal (powershell, también se puede realizar desde VCS) y ejecutar la siguiente instruccion:

```
$ ssh usuario@200.13.6.14 -p 10022
```

A continuación, se debe agregar la contraseña:

```
usuario@200.13.6.14's password:
```

Para cambiar la contraseña, se debe ejecutar:

```
[usuario@khipu ~] $ passwd
```

1.2.1 Conexión a través de WinSCP

Se debe descargar el software WinSCP desde el siguiente enlace: WinSCP

1.3 Cargando módulo Miniconda en la sesión

Una vez iniciada la sesión en Khipu, `env` no tendrá una ruta al ejecutable de `conda`. Para ejecutarlo, debe usar el siguiente comando.

```
$ module load conda/3-py39_4.12.0
```

Se puede ver los cambios de esta operación ejecutando el comando `env` antes y después de la instrucción anterior.

1.4 Creando un entorno para Python

Veamos la lista de entornos virtuales que tenemos instalados para nuestro usuario:

```
(base) ~ $ conda env list
```

Repliquemos en entorno Python del curso, con las librerias declaradas en el arvhivo `ima539KhipuEnv.yml`:

```yaml
name: ima539
dependencies:
  - ipykernel
  - python==3.9
  - pandas==1.4.4
  - scikit-learn==1.0.2
  - matplotlib==3.5.2
  - seaborn==0.11.2
```

Una vez que se tenga el archivo `ima539KhipuEnv.yml` en el directorio correspondiente, se puede hacer lo siguiente:

```
(base) ~ $ conda env create -f ima539KhipuEnv.yml
```

Luego de la instalación del entorno, se podrá acceder a él mediante:

```
(base) ~ $ conda activate ima539
```

Debe verse lo siguiente:

```
(ima539)  [usuario@khipu IMA539] $
```

Para volver al entorno base, se debe ejecutar:

```
(ima539)  [usuario@khipu IMA539] $ conda deactivate
```

1.5 Ejecutando un archivo.py en Khipu

Para ejecutar una rutina implementaba en base a un entorno Python, por ejemplo el entorno `ima539`, una vez activado el entorno, correr:

```
(ima539)  [usuario@khipu IMA539] $ python rutina.py
```

Por ejemplo, podemos ejecutar la rutina `Clase11_ejemplo1.py`, de la siguiente manera:

```
(ima539)  [usuario@khipu IMA539] $ python Clase11_ejemplo1.py
```

Nota importante: Para ejecutar un código Python en Khipu, este debe estar en formato `.py`. No está habilitado el servicio Jupyter Lab o Jupyter Notebook en este servidor.

[ ]:
```python
import numpy as np
import pandas as pd
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
from sklearn.metrics import matthews_corrcoef,cohen_kappa_score

df = pd.read_csv('https://archive.ics.uci.edu/ml/''machine-learning-databases/
 ↪iris/iris.data',header=None,encoding='utf-8')

y = df.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', -1, 1)

X = df.iloc[0:100, [0, 2]].values

X_std = np.copy(X)
X_std[:, 0] = (X[:, 0] - X[:, 0].mean()) / X[:, 0].std()
X_std[:, 1] = (X[:, 1] - X[:, 1].mean()) / X[:, 1].std()

def plot_decision_regions(X, y, classifier, resolution=0.02):
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
```

```
    for idx, cl in enumerate(np.unique(y)):
        if cl == -1:
            label = 'setosa'
        else:
            label = 'versicolor'
        plt.scatter(x=X[y == cl, 0],y=X[y == cl, 1],alpha=0.8,c=colors[idx],
                    marker=markers[idx], label=label,edgecolor='black')

class AdalineSGD(object):
    def __init__(self, eta=0.01, n_iter=10, shuffle=True, random_state=None):
        self.eta = eta
        self.n_iter = n_iter
        self.w_initialized = False
        self.shuffle = shuffle
        self.random_state = random_state

    def fit(self, X, y):
        self._initialize_weights(X.shape[1])
        self.cost_ = []
        for i in range(self.n_iter):
            if self.shuffle:
                X, y = self._shuffle(X, y)
            cost = []
            for xi, target in zip(X, y):
                cost.append(self._update_weights(xi, target))
            avg_cost = sum(cost) / len(y)
            self.cost_.append(avg_cost)
        return self

    def partial_fit(self, X, y):
        if not self.w_initialized:
            self._initialize_weights(X.shape[1])
        if y.ravel().shape[0] > 1:
            for xi, target in zip(X, y):
                self._update_weights(xi, target)
        else:
            self._update_weights(X, y)
        return self

    def _shuffle(self, X, y):
        r = self.rgen.permutation(len(y))
        return X[r], y[r]

    def _initialize_weights(self, m):
        self.rgen = np.random.RandomState(self.random_state)
        self.w_ = self.rgen.normal(loc=0.0, scale=0.01, size=1 + m)
```

```
        self.w_initialized = True

    def _update_weights(self, xi, target):
        output = self.activation(self.net_input(xi))
        error = (target - output)
        self.w_[1:] += self.eta * xi.dot(error)
        self.w_[0] += self.eta * error
        cost = 0.5 * error**2
        return cost

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def activation(self, X):
        return X

    def predict(self, X):
        return np.where(self.activation(self.net_input(X)) >= 0.0, 1, -1)

# Para entrenar 1 época
ada = AdalineSGD(n_iter=1, eta=0.08, random_state=1, shuffle=False)
ada.fit(X_std, y)

plot_decision_regions(X_std, y, classifier=ada)
plt.title('Adaline - Stochastic Gradient Descent')
plt.xlabel('largo sépalo [estandarizado]')
plt.ylabel('largo pétalo [estandarizado]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.savefig('ada_1.jpg', dpi=300)

y_pred = ada.predict(X_std)

confmat = confusion_matrix(y_true=y, y_pred=y_pred)

print('Matriz Confusión: \n', confmat)
print('Accuracy: %.3f' % accuracy_score(y_true=y, y_pred=y_pred))
print('Precision: %.3f' % precision_score(y_true=y, y_pred=y_pred))
print('Recall: %.3f' % recall_score(y_true=y, y_pred=y_pred))
print('F1: %.3f' % f1_score(y_true=y, y_pred=y_pred))
print('MCC: %.3f' % matthews_corrcoef(y_true=y, y_pred=y_pred))
print('Kappa: %.3f' % cohen_kappa_score(y1=y, y2=y_pred))
```

1.6 Trabajando a través de screen

Para realizar cómputos que conlleven un extenso periodo de ejecución, se recomienda trabajar con `screen`. Para esto se debe ejecutar en la terminal:

```
(ima539)  [usuario@khipu IMA539] $ screen -S name_screen
```

Una vez iniciada la `screen` podemos ejecutar la rutina `Clase11_ejemplo2.py`, dejarla corriendo por el tiempo que sea necesario, salir del servidor Khipu y volver luego, solo a chequear los resultados.

```
(ima539)  [usuario@khipu IMA539] $ python Clase11_ejemplo2.py
```

Para salir de la `screen` se debe ejecutar `CTRL+A CTRL+D`

Para volver a la `screen`, se debe ejecutar en la terminal:

```
(ima539)  [usuario@khipu IMA539] $ screen -r name_screen
```

[ ]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
from sklearn.metrics import matthews_corrcoef,cohen_kappa_score

df = pd.read_csv('https://archive.ics.uci.edu/ml/'
                 'machine-learning-databases'
                 '/breast-cancer-wisconsin/wdbc.data', header=None)

X = df.loc[:, 2:].values
y = df.loc[:, 1].values
le = LabelEncoder()
y = le.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.
 ↪20,stratify=y,random_state=1)

pipe_svc = make_pipeline(StandardScaler(),SVC(random_state=1))

param_range = [0.0001*0.5,0.0001,0.001*0.5,0.001,0.01*0.5,0.01,0.1*0.5,0.1,
               1.*0.5,1.,10.*0.5,10.,100.*0.5,100.,1000.*0.5,1000.]

param_grid = [{'svc__C': param_range,'svc__kernel': ['linear']},
              {'svc__C': param_range,'svc__gamma': param_range,'svc__kernel':␣
 ↪['rbf']},
              {'svc__C': param_range,'svc__gamma': param_range,'svc__kernel':␣
 ↪['poly']},
              {'svc__C': param_range,'svc__gamma': param_range,'svc__kernel':␣
 ↪['sigmoid']}]
```

```
gs =␣
  ↪GridSearchCV(estimator=pipe_svc,param_grid=param_grid,scoring='accuracy',cv=10,n_jobs=-1)
gs = gs.fit(X_train, y_train)

clf = gs.best_estimator_

print('Los mejores Hiperparámetros son: \n',gs.best_params_)

y_pred = clf.predict(X_test)

confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)

print('Matriz Confusión: \n', confmat)
print('Accuracy: %.3f' % accuracy_score(y_true=y_test, y_pred=y_pred))
print('Precision: %.3f' % precision_score(y_true=y_test, y_pred=y_pred))
print('Recall: %.3f' % recall_score(y_true=y_test, y_pred=y_pred))
print('F1: %.3f' % f1_score(y_true=y_test, y_pred=y_pred))
print('MCC: %.3f' % matthews_corrcoef(y_true=y_test, y_pred=y_pred))
print('Kappa: %.3f' % cohen_kappa_score(y1=y_test, y2=y_pred))
```

1.7 Volver a trabajar en Khipu

Para volver a trabajar en Khipu, los siguientes son los pasos que se deben realizar:

1. Protocolo `ssh` indicando usuario, servidor y puerto.

```
$ ssh usuario@200.13.6.14 -p 10022
```

2. Ingresar contraseña.

```
usuario@200.13.6.14's password:
```

3. Cargar módulo `conda`.

```
[usuario@khipu ~]$ module load conda/3-py39_4.12.0
```

4. Activar el entorno Python.

```
(base) [usuario@khipu ~]$ conda activate ima539
```

Debe verse lo siguiente:

```
(ima539)  [usuario@khipu ~] $
```

Está listo para computar!!