Python implementation of Decision Tree, Stochastic Gradient Descent, and Cross Validation

Balance Scale Data Set

- This data set was generated to model psychological experimental results.
- Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced.
- The attributes are the left weight, the left distance, the right weight, and the right distance.
- The correct way to find the class is the greater of (left-distance * left-weight) and (right-distance * right-weight).
- If they are equal, it is balanced.

Import libraries

- import numpy as np
- import pandas as pd
- from sklearn.cross_validation import train_test_split
- from sklearn.tree import DecisionTreeClassifier
- from sklearn.metrics import accuracy_score
- from sklearn import tree

Read from file

```
• data = pd.read_csv(
... 'http://archive.ics.uci.edu/ml/machine-learning-databases/balance-scale/balance-scale.data',
...                              sep= ',', header= None)
```

Print length of dataset

- `print('Dataset length:', len(name))`

```
Dataset length: 625
```

Data Slicing

- Dataset consists of 5 attributes
- 4 feature attributes and 1 target attribute
- The index of the target attribute is 1st

- X = data.values[:,1:5]
- Y = data.values[:,0]

Split dataset between train and test

- X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3)

- X_train and y_train = training data
- X_test and y_test = test data

- Test_size = test set will be 30% of whole dataset and training will be 70%

Decision Tree Training

- clf_entropy = DecisionTreeClassifier(max_depth=3)
- clf_entropy.fit(X_train, y_train)

- **Result**
- DecisionTreeClassifier(compute_importances=None, criterion='gini',

```
max_depth=3, max_features=None, max_leaf_nodes=None,
min_density=None, min_samples_leaf=1, min_samples_split=2,
random_state=None, splitter='best')
```

Prediction

- y_pred_en = clf_entropy.predict(X_test)
- y_pred_en

Stochastic Gradient Descent

- from sklearn.linear_model import SGDClassifier

```
X = [[0., 0.], [1., 1.]]
y = [0, 1]
clf = SGDClassifier(loss="hinge", penalty="l2")
clf.fit(X, y)
```

- Predict new values

```
clf.predict([[2., 2.]])
```

Cross Validation

- import numpy as np
- from sklearn.cross_validation import KFold
- X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
- y = np.array([1, 2, 3, 4])
- kf = KFold(4, n_folds=2)
- len(kf)
- 2
- print(kf)
- sklearn.cross_validation.KFold(n=4, n_folds=2, shuffle=False, random_state=None)

Cross Validation

```
for train_index, test_index in kf:
...     print('TRAIN:', train_index, 'test:', test_index)
...
 TRAIN: [2 3] test: [0 1]
 TRAIN: [0 1] test: [2 3]
```

END