AI4FutureWorkForce Data Processing

Exploration of predicting course completion using sklearn and Tensorflow

Dependencies

These are all the libraries we need to run this notebook

```python
import pandas as pd
import plotly.offline as plt
import _pickle as pickle
import os
import numpy as np

import tensorflow as tf

import tensorflow.keras as keras

import plotly.graph_objs as go
import itertools

from sklearn.naive_bayes import GaussianNB
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn import linear_model
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from collections import defaultdict
from sklearn.metrics import classification_report
from numpy.random import seed
seed(42)
from tensorflow import set_random_seed
set_random_seed(42)
from tensorflow.python.client import device_lib

plt.init_notebook_mode(connected=True)
```

```python
# Verify that the TF device we're using is what we expect
device_lib.list_local_devices()
```

```
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 5457231463178488201, name: "/device:XLA_CPU:0"
 device_type: "XLA_CPU"
 memory_limit: 17179869184
 locality {
 }
 incarnation: 7095777417938979866
 physical_device_desc: "device: XLA_CPU device"]
```

Load Data

```python
# Whether or not to use raw or processed dfs
PREPROCESS = 0
# Whether to fill fee columns or just encode as NaN
FILL = 0
SHUFFLE = True

NUM_CLASSES = 2
```

```python
# List all columns of the tables in data
column_list = ['Age (Birthday Masked)','Income','Education',
               'MAX(Learner Test Score)','Primary Interest In Course',
               'Hours Coded','How Many Hours A Week Can You Commit To Class',
               'Promise Zone Indicator','Hacker Rank Score', 'Location; number']

# List desired columns for train/test/validation
desired_columns = ['Age (Birthday Masked)','Income','Education',
               'MAX(Learner Test Score)','Primary Interest In Course',
               'Hours Coded','How Many Hours A Week Can You Commit To Class',
               'Promise Zone Indicator','Hacker Rank Score', 'Location; number']

# Load raw data and run pre-processing
if PREPROCESS == 1:
    df_data = pickle.load( open( "df_data.p", "rb" ) )
    df_labels = pickle.load( open( "df_labels.p", "rb" ) )

    df_validata = pickle.load( open( "df_validata.p", "rb" ) )
    df_valilabels = pickle.load( open( "df_valilabels.p", "rb" ) )

    df_valilabels = df_valilabels.add(1)
    df_labels = df_labels.add(1)

    X_train, X_test, X_val, y_train, y_test, y_val = data_prepare.prepare
    (column_list, desired_columns, df_data, df_validata, df_labels,
     df_valilabels, SHUFFLE, FILL)

    print(X_train.shape, X_test.shape, X_val.shape)

# Otherwise load pre-processed data
else:
    X_train = pickle.load( open( "X_train.p", "rb" ) )
    X_test = pickle.load( open( "X_test.p", "rb" ) )
    X_val = pickle.load( open( "X_val.p", "rb" ) )

    y_train = pickle.load( open( "y_train.p", "rb" ) )
    y_test = pickle.load( open( "y_test.p", "rb" ) )
    y_val = pickle.load( open( "y_val.p", "rb" ) )
```

Checking that the dataframe looks like it should:

```python
X_train.head()
```

	Age (Birthday Masked)	Income	MAX(Learner Test Score)	Hours Coded	How Many Hours A Week Can You Commit To Class	Promise Zone Indicator	Location; number
1341	0.353798	1.832671	0.988618	-0.055229	-2.027239	-0.405866	0.571825
1649	-1.263828	-0.299421	0.018359	0.959409	0.381102	-0.405866	0.571825
998	-0.940303	-1.152258	1.312038	0.959409	0.381102	-0.405866	0.571825

```
y_train
```

```
array([1, 1, 1, ..., 1, 0, 1])
```

Classification Functions

```
class EpochTrack(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        if epoch % 100 == 0:
            print('')
        print('.', end='')


def eval_model(model, test_data, test_labels, VERBOSE, NUM_CLASSES):
    test_labels_sparse = keras.utils.to_categorical(test_labels, NUM_CLASSES)
    [loss, mae, acc] = model.evaluate(test_data, test_labels_sparse,
                                      verbose=VERBOSE)

    print("Mean Abs Error:\t{:7.2f}".format(mae * 1000))
    print("Loss:\t\t", loss)
    print("Accuracy:\t", acc)

    preds = model.predict(test_data)
    y_preds = preds.argmax(axis=-1)
    print(classification_report(test_labels, y_preds))

    return acc, preds


def plot_history(history):
    # Define each data series
    trace1 = go.Scatter(x=history.epoch, y=np.array
                        (history.history['mean_absolute_error']),
                        name='Training Loss')
    trace2 = go.Scatter(x=history.epoch, y=np.array
                        (history.history['val_mean_absolute_error']),
                        name='Val Loss')

    # Add each series
    data = [trace1, trace2]

    # Define graph layout
    layout = go.Layout(
        title='Training History',
        xaxis=dict(title='Epoch'),
        yaxis=dict(title='Mean Abs Error'))

    fig = go.Figure(data=data, layout=layout)
    plt.iplot(fig)

def show_incorrect(y_classes, data, act):
    incorrects = np.nonzero(y_classes.reshape((-1,)) != act)
    test_X = data.iloc[incorrects]

    test_y = pd.DataFrame(y_classes[tuple(incorrects)])
    test_y.columns = ['Prediction']

    act_y = pd.DataFrame(act[tuple(incorrects)])
    act_y.columns = ['Actual']

    test_X.index = range(len(test_X))

    frames = [test_X, test_y, act_y]
    df_pred = pd.concat(frames, axis=1)

    return df_pred


def skl_evaluate(model, desired_columns, X, y):
    score = model.score(X, y.astype(int))

    preds = model.predict(X)
    print(classification_report(y, preds))

    print("Accuracy on dataset:\t %f\n" % score)

    importances = model.feature_importances_

    indices = np.argsort(importances)[::-1]

    # Print the feature ranking
    print("Feature ranking:")
    for f in range(X.shape[1]):
        print("%d. %s \t(%f)" % (f + 1, desired_columns[indices[f]], importances[indi

    return score
```

Classify

```
nb = GaussianNB()
nb.fit(X_train, y_train.astype(int))
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
X_test.iloc[0, :]
```

```
Age (Birthday Masked)                            -1.048144
Income                                            0.553416
MAX(Learner Test Score)                           0.341779
Hours Coded                                      -0.055229
How Many Hours A Week Can You Commit To Class     1.585273
Promise Zone Indicator                           -0.405866
Location; number                                  0.571825
Name: 522, dtype: float64
```

```
nb_score = nb.score(X_test, y_test.astype(int))
print("Naive Bayes accuracy on test:\t %f" % nb_score)

nb_score_val = nb.score(X_val, y_val.astype(int))
```

```
Naive Bayes accuracy on test:     0.568765
```

SVM

```
clf = svm.SVC(kernel='linear', C=1, random_state=42)
clf.fit(X_train, y_train.astype(int))
```

```
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=42,
    shrinking=True, tol=0.001, verbose=False)
```

```
clf_score = clf.score(X_test, y_test.astype(int))
print("SVM accuracy on test:\t %f" % clf_score)

```

```
clf_score_val = clf.score(X_val, y_val.astype(int))
```

```
SVM accuracy on test:     0.596737
```

Linear Regression

```
linr = linear_model.LinearRegression(n_jobs = -1)
linr.fit(X_train, y_train.astype(int))
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=False)
```

```
linr_score = linr.score(X_test, y_test.astype(int))
print("Linear Regression coefficient of determination, 1 is perfect prediction:\t %f"

linr_score_val = linr.score(X_val, y_val.astype(int))
```

```
Linear Regression coefficient of determination, 1 is perfect prediction:         0.
```

Random Forest

```
rf = RandomForestClassifier(criterion='gini', max_depth=5,
                               min_samples_leaf=5, min_samples_split=2,
                               n_estimators = 220, oob_score=True,
                               max_features=0.5, n_jobs = -1, random_state=42)

rf.fit(X_train, y_train.astype(int))
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=5, max_features=0.5, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=220,
                       n_jobs=-1, oob_score=True, random_state=42, verbose=0,
                       warm_start=False)
```

```
print("Random Forest - Test set")
rf_score = skl_evaluate(rf, desired_columns, X_test, y_test)

print("Random Forest - Validation set")
rf_score_val = skl_evaluate(rf, desired_columns, X_val, y_val)
```

```
Random Forest - Test set
              precision    recall  f1-score   support

           0       0.65      0.79      0.72       241
           1       0.64      0.46      0.54       188

    accuracy                           0.65       429
   macro avg       0.64      0.63      0.63       429
weighted avg       0.65      0.65      0.64       429

Accuracy on dataset:     0.648019

Feature ranking:
1. Education    (0.275820)
2. Age (Birthday Masked)        (0.238993)
3. MAX(Learner Test Score)      (0.222282)
4. Income       (0.123961)
5. Primary Interest In Course   (0.076364)
6. How Many Hours A Week Can You Commit To Class        (0.034888)
7. Hours Coded  (0.027692)
Random Forest - Validation set
              precision    recall  f1-score   support

           0       0.78      0.85      0.82       308
           1       0.51      0.39      0.44       121
```

Random Forest Gridsearch Optimisation

```
weighted avg       0.70      0.72      0.71       429
```

```python
rf=RandomForestClassifier(random_state=42)

param_grid = {
    'n_estimators': [100, 120, 140, 160, 180, 200, 220, 240, 260],
    'min_samples_leaf': [3, 5, 7],
    'min_samples_split': [2, 3, 4, 5, 6],
    'max_depth' : [5, 10, 15, 20, 25, 30, 35, 40, 45],
    'criterion' :['gini', 'entropy']
}


CV_rf = GridSearchCV(estimator=rf, param_grid=param_grid, cv= 5, n_jobs = -1,
                     verbose = 2)
CV_rf.fit(X_train, y_train.astype(int))
CV_rf.best_params_
```

This is sample output from this step:

```
Fitting 5 folds for each of 2430 candidates, totalling 12150 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:    8.8s
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:   33.4s
[Parallel(n_jobs=-1)]: Done 361 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 644 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done 1009 tasks      | elapsed:  3.7min
[Parallel(n_jobs=-1)]: Done 1454 tasks      | elapsed:  5.5min
[Parallel(n_jobs=-1)]: Done 1981 tasks      | elapsed:  7.7min
[Parallel(n_jobs=-1)]: Done 2588 tasks      | elapsed: 10.2min
[Parallel(n_jobs=-1)]: Done 3277 tasks      | elapsed: 13.0min
[Parallel(n_jobs=-1)]: Done 4046 tasks      | elapsed: 16.2min
```

```
[Parallel(n_jobs=-1)]: Done 4897 tasks      | elapsed: 19.7min
[Parallel(n_jobs=-1)]: Done 5828 tasks      | elapsed: 23.5min
[Parallel(n_jobs=-1)]: Done 6841 tasks      | elapsed: 27.4min
[Parallel(n_jobs=-1)]: Done 7934 tasks      | elapsed: 32.2min
[Parallel(n_jobs=-1)]: Done 9109 tasks      | elapsed: 37.4min
[Parallel(n_jobs=-1)]: Done 10364 tasks      | elapsed: 43.0min
[Parallel(n_jobs=-1)]: Done 11701 tasks      | elapsed: 49.0min
[Parallel(n_jobs=-1)]: Done 12150 out of 12150 | elapsed: 50.9min finished
```

The optimum parameters are: {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 5, 'min_samples_split': 2, 'n_estimators': 220}

```python
CV_rf_score = CV_rf.score(X_test, y_test.astype(int))
print("Optimised Random Forest accuracy on test:\t %f" % CV_rf_score)
```

Random Forest - Find Optimal Parameters

```python
def skl_run_combos(combo_columns, model, df_data, df_labels, SHUFFLE, FILL):

    cool_list = list()

    X_train_combo = X_train
    X_test_combo = X_test

    y_train_combo = y_train
    y_test_combo = y_test

    for L in range(0, len(combo_columns) + 1):
        combinations = list(itertools.combinations(combo_columns, L))[1:]
        for combo in combinations:
            combo = list(combo)

            model.fit(X_train_combo[combo], y_train)
            rf_score = model.score(X_test_combo[combo], y_test)
            output = str(rf_score) + " " + str(combo)
            print(output)
            cool_list.append(output)

    cool_list.sort()

    with open('outputs.txt', 'w') as f:
        for item in cool_list:
            f.write("%s\n" % item)


def normalise_df(train_df, test_df, AXIS, val_df=1):
    mu = train_df.mean(axis=AXIS)
    sd = train_df.std(axis=AXIS)

    train_df = (train_df - mu) / sd
    test_df = (test_df - mu) / sd
    val_df = (val_df - mu) / sd

    return train_df, test_df, val_df
```

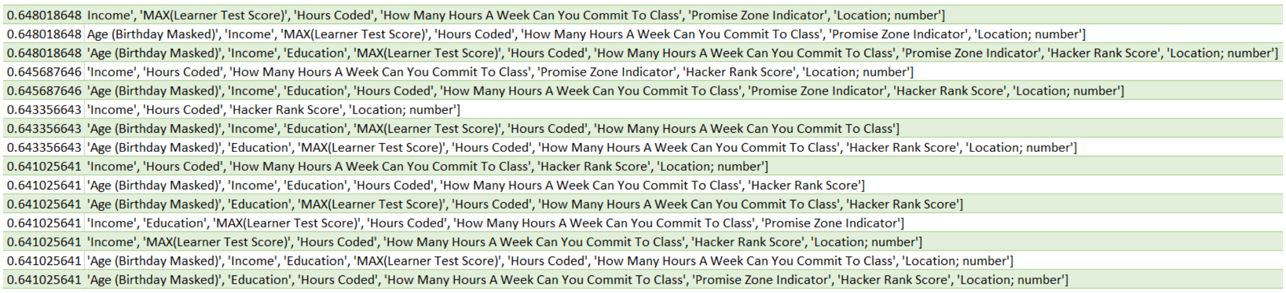
```python
df_data = pickle.load( open( "df_data.p", "rb" ) )
df_labels = pickle.load( open( "df_labels.p", "rb" ) )

combo_columns = ['Age (Birthday Masked)','Income','Education',
                 'MAX(Learner Test Score)','Primary Interest In Course',
                 'Hours Coded','How Many Hours A Week Can You Commit To Class',
                 'Promise Zone Indicator','Hacker Rank Score',
                 'Location; number']

model = RandomForestClassifier(criterion='gini', max_depth=5,
                               min_samples_leaf=5, min_samples_split=2,
                               n_estimators = 220, oob_score=True,
                               max_features=0.5, n_jobs = -1, random_state=42)

print('Accuracy on test, [columns used]')
skl_run_combos(combo_columns, model, df_data, df_labels,
                            SHUFFLE, FILL)
```

This is a sample output from this step:



```
0.648018648 Income', 'MAX(Learner Test Score)', 'Hours Coded', 'How Many Hours A Week Can You Commit To Class', 'Promise Zone Indicator', 'Location; number']
0.648018648 Age (Birthday Masked)', 'Income', 'MAX(Learner Test Score)', 'Hours Coded', 'How Many Hours A Week Can You Commit To Class', 'Promise Zone Indicator', 'Location; number']
0.648018648 'Age (Birthday Masked)', 'Income', 'Education', 'MAX(Learner Test Score)', 'Hours Coded', 'How Many Hours A Week Can You Commit To Class', 'Promise Zone Indicator', 'Hacker Rank Score', 'Location; number']
0.645687646 'Income', 'Hours Coded', 'How Many Hours A Week Can You Commit To Class', 'Promise Zone Indicator', 'Hacker Rank Score', 'Location; number']
0.645687646 'Age (Birthday Masked)', 'Income', 'Education', 'Hours Coded', 'How Many Hours A Week Can You Commit To Class', 'Promise Zone Indicator', 'Hacker Rank Score', 'Location; number']
0.643356643 'Income', 'Hours Coded', 'Hacker Rank Score', 'Location; number']
0.643356643 'Age (Birthday Masked)', 'Income', 'Education', 'MAX(Learner Test Score)', 'Hours Coded', 'How Many Hours A Week Can You Commit To Class']
0.643356643 'Age (Birthday Masked)', 'Education', 'MAX(Learner Test Score)', 'Hours Coded', 'How Many Hours A Week Can You Commit To Class', 'Hacker Rank Score', 'Location; number']
0.641025641 'Income', 'Hours Coded', 'How Many Hours A Week Can You Commit To Class', 'Hacker Rank Score', 'Location; number']
0.641025641 'Age (Birthday Masked)', 'Income', 'Education', 'Hours Coded', 'How Many Hours A Week Can You Commit To Class', 'Hacker Rank Score']
0.641025641 'Age (Birthday Masked)', 'Education', 'MAX(Learner Test Score)', 'Hours Coded', 'How Many Hours A Week Can You Commit To Class', 'Hacker Rank Score']
0.641025641 'Income', 'Education', 'MAX(Learner Test Score)', 'Hours Coded', 'How Many Hours A Week Can You Commit To Class', 'Promise Zone Indicator']
0.641025641 'Income', 'MAX(Learner Test Score)', 'Hours Coded', 'How Many Hours A Week Can You Commit To Class', 'Hacker Rank Score', 'Location; number']
0.641025641 'Age (Birthday Masked)', 'Income', 'Education', 'MAX(Learner Test Score)', 'Hours Coded', 'How Many Hours A Week Can You Commit To Class', 'Location; number']
0.641025641 'Age (Birthday Masked)', 'Education', 'Hours Coded', 'How Many Hours A Week Can You Commit To Class', 'Promise Zone Indicator', 'Hacker Rank Score', 'Location; number']
```

Multi Layer Perceptron

```
def build_mlp(input, NUM_CLASSES):
    model = keras.Sequential([
        keras.layers.Dense(8, activation='relu', input_shape=(input.shape[1],)),
        keras.layers.Dense(5, activation='relu'),
#         keras.layers.Dense(3, activation='relu'),
        keras.layers.Dense(NUM_CLASSES, activation='softmax')
    ])

    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['mae', 'acc'])

    model.summary()

    return model


def train_mlp(dataframe, labels, model, checkpoint_path, EPOCHS):
    # Limit the training when there are multiple epochs with little change loss
    # The patience parameter is the amount of epochs to check for improvement
    early_stop = keras.callbacks.EarlyStopping(monitor='val_loss',
                                               patience=4000)

    # Create checkpoint callback
    cp_callback = keras.callbacks.ModelCheckpoint(checkpoint_path,
                                                  save_weights_only=True,
                                                  verbose=0)

    # Track the training statistics
    history = model.fit(dataframe, labels, epochs=EPOCHS,
                        validation_split=0.2, verbose=0,
                        callbacks=[early_stop, EpochTrack(), cp_callback])

    print("\nEpochs: {}".format(len(history.epoch)))
    plot_history(history)

    return model
```

```
y_train_sparse = keras.utils.to_categorical(y_train, NUM_CLASSES)
```

```

TRAIN = 0

if TRAIN == 1:
    checkpoint_path = "training_1/cp.ckpt"
    checkpoint_dir = os.path.dirname(checkpoint_path)

    # Train MLP and save to checkpoints
    %time mlp = train_mlp(X_train, y_train_sparse, build_mlp(X_train, NUM_CLASSES), cl

else:
    checkpoint_path = "training_1/cp.ckpt"

    checkpoint_dir = os.path.dirname(checkpoint_path)

    # Load trained weights from the checkpoint path
    mlp = build_mlp(X_train, NUM_CLASSES)
    mlp.load_weights(checkpoint_path)

# Test
print("\nMLP - Test set")
acc_test, preds_test = eval_model(mlp, X_test, y_test, 0, NUM_CLASSES)

print("\nMLP - Validation set")
acc_val, preds_val = eval_model(mlp, X_val, y_val, 0, NUM_CLASSES)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/op
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

MLP every combination of parameters

```
dense_1 (Dense)              (None, 5)                 45
```

```python
col_names = ['Age (Birthday Masked)','Income','Education',
             'MAX(Learner Test Score)','Primary Interest In Course',
             'Hours Coded','How Many Hours A Week Can You Commit To Class'
             ,'Promise Zone Indicator','Hacker Rank Score','Location; number']

cool_list = list()

y_train_sparse = keras.utils.to_categorical(y_train, NUM_CLASSES)
y_val_sparse = keras.utils.to_categorical(y_val, NUM_CLASSES)

# with open('results.txt', 'w') as f:
#   for L in range(0, len(col_names)+1):
#       combinations = list(itertools.combinations(col_names, L))[4:]
#       for combo in combinations:
#           combo = list(combo)

#           mlp = train_mlp(X_train[combo], y_train_sparse,
#                           build_mlp(X_train[combo], NUM_CLASSES),
#                           checkpoint_path, EPOCHS=1000)
#           acc_val = eval_model(mlp, X_val[combo], y_val_sparse, 0, NUM_CLASSES)

#           output = str(acc_val) + " " + str(combo) + "\n"
#           print(output)
#           cool_list.append(output)
#           f.write(str(cool_list))
```

```
MLP - Validation set
```

Test Results Summary

```
              precision    recall  f1-score   support
```

```python
print("Naive Bayes accuracy on validation:\t\t %f" % nb_score_val)
print('')
print("SVM accuracy on validation:\t\t\t %f" % clf_score_val)
print('')
print("Linear Regression accuracy on validation:\t %f" % linr_score_val)
print('')
print("Random Forest accuracy on validation:\t\t %f" % rf_score_val)
print('')
print("MLP accuracy on validation:\t\t\t %f" % acc_val)
```

```
Naive Bayes accuracy on validation:              0.641026

SVM accuracy on validation:                      0.727273

Linear Regression accuracy on validation:        0.051558

Random Forest accuracy on validation:            0.722611

MLP accuracy on validation:                      0.668998
```

Insight

Let's sanity check the results by comparing model predictions to the real test results.

```python
predictions = preds_test

y_classes = predictions.argmax(axis=-1)
```

When comparing predictions to true labels we'd expect to see mostly true values for high scoring classifiers

```python
y_classes = y_classes.flatten()

# Match is 'True' when values are the same
match = [y_classes[x] == y_test[x] for x in range(len(y_classes))]

# Combine and show
df_compare = pd.DataFrame({'Predictions': y_classes, 'Actual': y_test,
                           'Match': match})

print('Show performance on first 5 rows')
df_compare.head(5)
```

```
Show performance on first 5 rows
```

	Predictions	Actual	Match
0	1	1	True
1	0	0	True
2	1	1	True
3	0	1	False
4	1	1	True

We can also obtain the set of rows that were incorrectly predicted

```python
df_pred = show_incorrect(y_classes, X_test, y_test)

df_pred.sort_values('Prediction', inplace=True)
print('Show first 5 rows that had wrong predictions')
print('Values are encoded as per earlier steps')
df_pred.head(5)
```

```
Show first 5 rows that had wrong predictions
Values are encoded as per earlier steps
```

	Age (Birthday Masked)	Income	MAX(Learner Test Score)	Hours Coded	How Many Hours A Week Can You Commit To Class	Promise Zone Indicator	Location; number
0	1.647899	1.832671	-0.628481	-0.055229	0.381102	-0.405866	0.571825
125	-0.185411	-1.578677	0.988618	-0.055229	-0.823068	-0.405866	0.571825
124	-0.293252	-0.299421	0.341779	1.974047	0.381102	-0.405866	0.571825