Regression

Master IoT

C. Guyeux

From data to dataframe

From csv:

```
pd.read_csv(my_csv, sep=";", dtype={"DeptNum": str})
```

From excel:

```
pd.read_excel(my_excel)
```

From python dictionary:

```
pd.DataFrame.from_dict(my_dict, orient="index")
```

```
from pyarrow import feather
df = feather.read_feather(fic)
```

It is better to specify the type of columns in a DataFrame. Because if Pandas doesn't know the type, it will scan each column to see what type to associate with it (dataframe in memory).

Save a DataFrame

DataFrame -> csv:

```
df.to_csv('my_file.csv', index=False)
```

with feather:

```
from pyarrow import feather
feather.write_feather(df, 'my_file.feather')
```

Detecting with NaN

Count the NaNs per column:

```
for k in sorted(df.columns):
    if df[k].isna().sum() > 0:
        print(k, df[k].isna().sum())
```

The indices with NaN:

```
for k in sorted(features.columns):
    index = features[k].index[features[k].apply(np.isnan)]
```

Filling NaN

```
df.fillna(method='ffill', inplace=True)
```

ffill: propagate last valid observation forward to next valid
bfill: use next valid observation to fill gap.

```
df.interpolate()
```

linearly interpolated values based on adjacent values

```
df.interpolate(method='polynomial', order=2)
```

Managing particular columns

Count the number of constant columns (where the number of unique values is 1):

```
(df.nunique() == 1).sum()
```

Remove the constant columns:

```
df = df.loc[:, (df != df.iloc[0]).any()]
```

Find duplicated indices:

```
df.index.duplicated().sum()
```

Drop them:

```
df.drop_duplicates(subset=['col1, 'col2], keep="first")
```

Preparing the data: digital ones

Standardising digital data:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df[num_cols] = scaler.fit_transform(df[num_cols])
```

Preparing the data: categorical ones

One-hot encoding: creates a new binary column for each unique category in the original column.

```
pd.get_dummies(df, columns=['A'])
```

Label encoding: assigns a unique integer to each category in the original column.

```
from sklearn.preprocessing import LabelEncoder
df['A_encoded'] = LabelEncoder().fit_transform(df['A'])
```

Target encoding: replacing the original categorical values with their corresponding calculated means.

```
from category_encoders import TargetEncoder
enc = TargetEncoder(cols=cat_cols)
encoder = enc.fit(df.drop(target, axis=1), df[target])
target_enc = encoder.transform(df.drop(target, axis=1))
```

Feature selection

Delete low variance columns:

```
variances = df.var()
high_variance_columns = variances[variances > 0.5].index
df = df[high_variance_columns]
```

Find columns that are highly correlated with target:

```
corr = df.corr()
target_corr = corr['target']
corr_threshold = 0.7
columns = target_corr[target_corr.abs()>corr_threshold].index
```

etc.

Train and test datasets (with validation)

```
from sklearn.model_selection import train_test_split

train_val_set, test_set = train_test_split(df, test_size=0.2,
                                           random_state=42)
train_set, val_set = train_test_split(train_val_set, test_size=0.2,
                                      random_state=42)

X_train = train_set.drop(target, axis=1)
X_val = val_set.drop(target, axis=1)
X_test = test_set.drop(target, axis=1)

y_train = train_set[target]
y_val = val_set[target]
y_test = test_set[target]
```

Learning (XGBoost)

```
reg = xgboost.XGBRegressor(random_state = random_state,
                           early_stopping_rounds = 15,
                           learning_rate=0.1,
                           max_depth = 7,
                           n_estimators=100000,
                           min_child_weight=1,
                           gamma=0,
                           subsample=0.8,
                           colsample_bytree=0.8,
                           n_jobs=-1,
                           verbosity=0)

reg.fit(X_train, y_train, eval_set = [(X_val, y_val)])
```

Prediction - evaluation (XGBoost)

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

y_pred = reg.predict(X_test)
print(f"MAE : {mean_absolute_error(y_test, y_pred)}")
print(f"MSE : {mean_squared_error(y_test, y_pred)}")
```

Improve the prediction score

- Cross-validation
- Play with hyperparameters
  - Manually
  - Grid search
  - AutoML
- Change the model : LightGBM, Random forests, SVM, ADABoost…
- Add new features
- Better feature selection
- etc.

Graded projects

1. Clustering project (1/4 of the final mark)
https://cours-info.iut-bm.univ-fcomte.fr/pmwiki-2.2.131/pmwiki.php/IoTen/ClusteringProject
2. Classification/regression project (1/4 of the final mark)
https://cours-info.iut-bm.univ-fcomte.fr/pmwiki-2.2.131/pmwiki.php/IoT/RegressionProject

The jupyter notebook is to be sent, as a pdf, by the end of this week (deadline: Sunday 9 April midnight) to: cguyeux@femto-st.fr

There will also be an exam later (you will be notified in advance).