DHRUV GARG 16 BCE 1190

PIPELINING AND GRID SEARCH

LAB 11

Pipelining theory:

The main idea behind pipeline utility is to help automate machine learning workflows. Pipelines work by allowing a linear sequence of data transforms to be chained together culminating in a modelling process that can be evaluated. It provides a mechanism to construct a multi-ML parallel pipeline system to compare the results of several ML methods. **Each stage of a pipeline is fed data processed from its preceding stage; that is, the output of a processing unit is supplied as the input to the next step.** The data flows through the pipeline just as water flows in a pipe.

Thus, we are able to ensure that all of the steps in the pipeline are constrained to the data available for the evaluation, such as the training dataset or each fold of the cross validation procedure.

Grid Search theory:

Grid search in basic sense, is a brute force method to estimate **hyperparameters**. If we have **k** hyperparameters, and each one of them have **ci** possible values. Depending on the type of model utilized, certain parameters are necessary. Then, performing grid search is basically taking a Cartesian product of these possible values. Therefore,

Total number of possibilities: $\prod_{i=1}^{k} c_i$

Grid search is really time consuming, but we can run all the possible combinations in parallel; since they are embarrassingly parallel. Also, grid-searching **does not apply to only one model type**. It can be applied across machine learning to calculate the best parameters to use for any given model.

PART 1: Lab: IRIS Dataset (CLASSIFICATION)

We implemented various classification algorithms on the Iris dataset. Since pipelines are used for data processing, we first created a classifier pipeline with algorithms like logistic regression, support vector machine and decision tree classifier (with default parameters). All the 3 had the data scaled, and the same random_state = 42. To check which model classified the best, the training data was fit to all the 3 algorithms. Since these are classification models, **accuracy_score metric** was utilized. The best accuracy was given by logistic regression (93.3%). As the first grid search implementation, we applied **grid search on decision tree** on a given set of parameters. On completion, the **tuned hyperparameters** were also displayed – criterion: gini index, max_depth: 3, min_samples_leaf: 1 and min_samples_split: 2. We note that the best parameters given by grid search were used to test the accuracy on the testing data.

To integrate pipeline with grid search, the **grid search parameters for each algorithm were separately defined**. The pipeline was re-run for the 3 models (with and without PCA), along with grid-search. The hyperparameters for logistic regression, random forest and support vector machines were separately defined as **grid_params arrays**. Although this was a time consuming process, we were sure that the **best possible tuning of models** had been achieved in the process. In this case**, Random forest** with criterion-entropy and max_depth: 3 gave the best accuracy on test data, which was (100%). **Interestingly, the 3 models with PCA, gave a lower accuracy on test data than their non-PCA models.**

PART 2: ABSENTEEISM Dataset (REGRESSION)

On the Absenteeism at work dataset, two regression algorithms were applied to predict the number of hours the employee was absent. The two algorithms were: logistic regression for regression and support vector machine for regression. Since this problem is on regression, two different metrics were used: **score (higher is better) and mean_squared_error (lower is better).** First, similar to the Iris dataset implementation, a pipeline of the 2 models (with default parameters) was constructed. When the data was fit to the 2 models, it was found that the **logistic regression** model gave the highest score (0.297), and was the best model. **SVR** gave a bad score of -0.053. Next, a simple grid search for **support vector regression** was implemented, which **gave the best parameters** as- C:5, gamma:10, kernel: rbf and tol: 0.001.

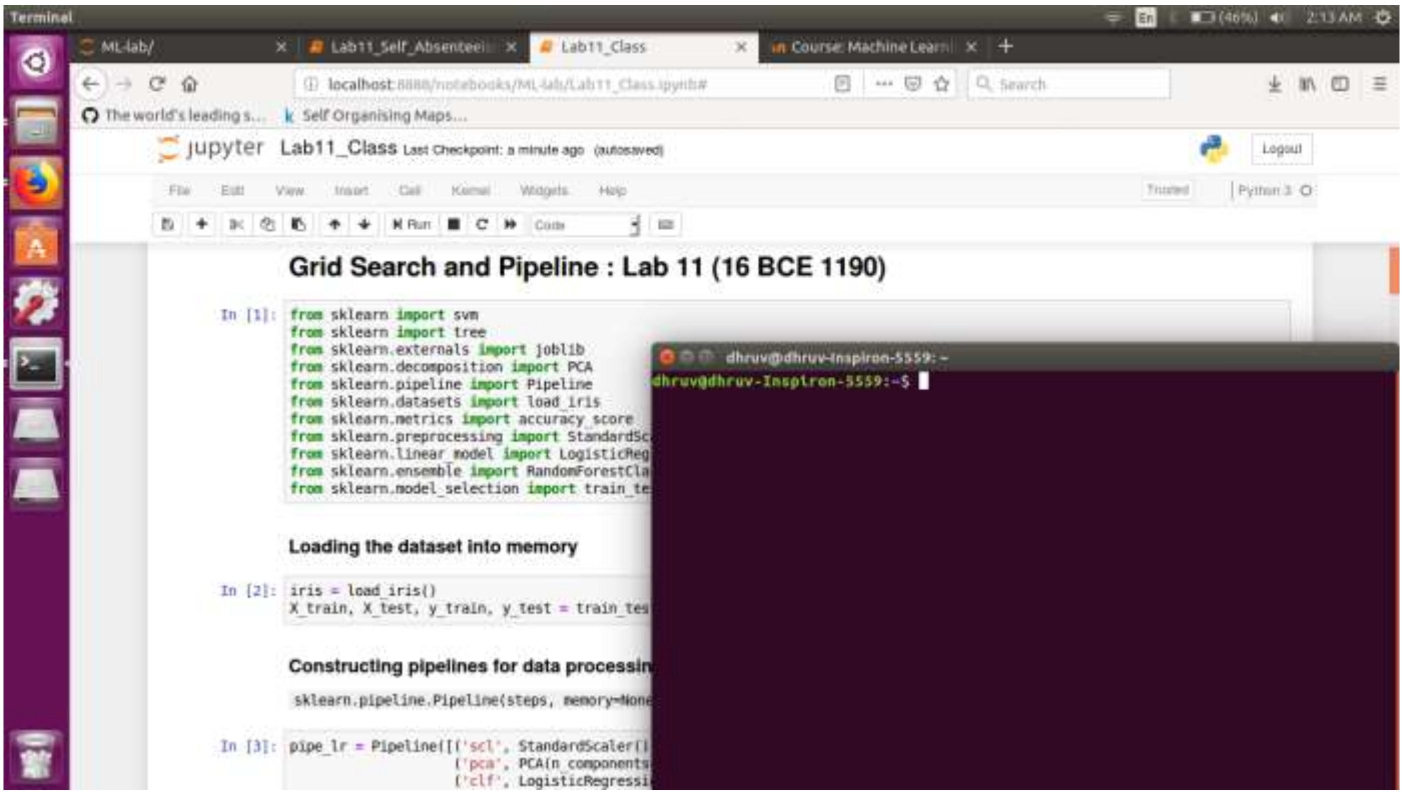
Finally, a pipeline for the two models was implemented, along with grid search. The **hyperparameters** for logistic regression and support vector regression were separately defined as **grid_params arrays**. Although this was a time consuming process, we were sure that the best possible tuning of models had been achieved in the process. The best parameters for each of the 2 regression models was displayed as a result from grid search. For **logistic regression**, they were- penalty: 'L2' and solver: 'lbfgs'. It gave a mean_squared_error of 79.45. **Support Vector Regressor** was the **best model**, with a slightly lower mean_squared_error at 76.51. Its tuned hyperparameters were- C:5, and kernel: rbf.

CONCLUSION

Thus, GridSearchCV and pipelining was successfully implemented for both classification and regression.

SCREENSHOTS

Lab implementation


Terminal
ML-lab/
Lab11_Self_Absenteei
Lab11_Class
Course: Machine Learni
localhost:8888/notebooks/ML-lab/Lab11_Class.ipynb#
The world's leading s...
Self Organising Maps...
jupyter Lab11_Class Last Checkpoint: a minute ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
dhruv@dhruv-Inspiron-5559: ~

Grid Search and Pipeline : Lab 11 (16 BCE 1190)

```
In [1]: from sklearn import svm
        from sklearn import tree
        from sklearn.externals import joblib
        from sklearn.decomposition import PCA
        from sklearn.pipeline import Pipeline
        from sklearn.datasets import load_iris
        from sklearn.metrics import accuracy_score
        from sklearn.preprocessing import StandardSc
        from sklearn.linear_model import LogisticReg
        from sklearn.ensemble import RandomForestCla
        from sklearn.model_selection import train_te
```

Loading the dataset into memory

```
In [2]: iris = load_iris()
        X_train, X_test, y_train, y_test = train_tes
```

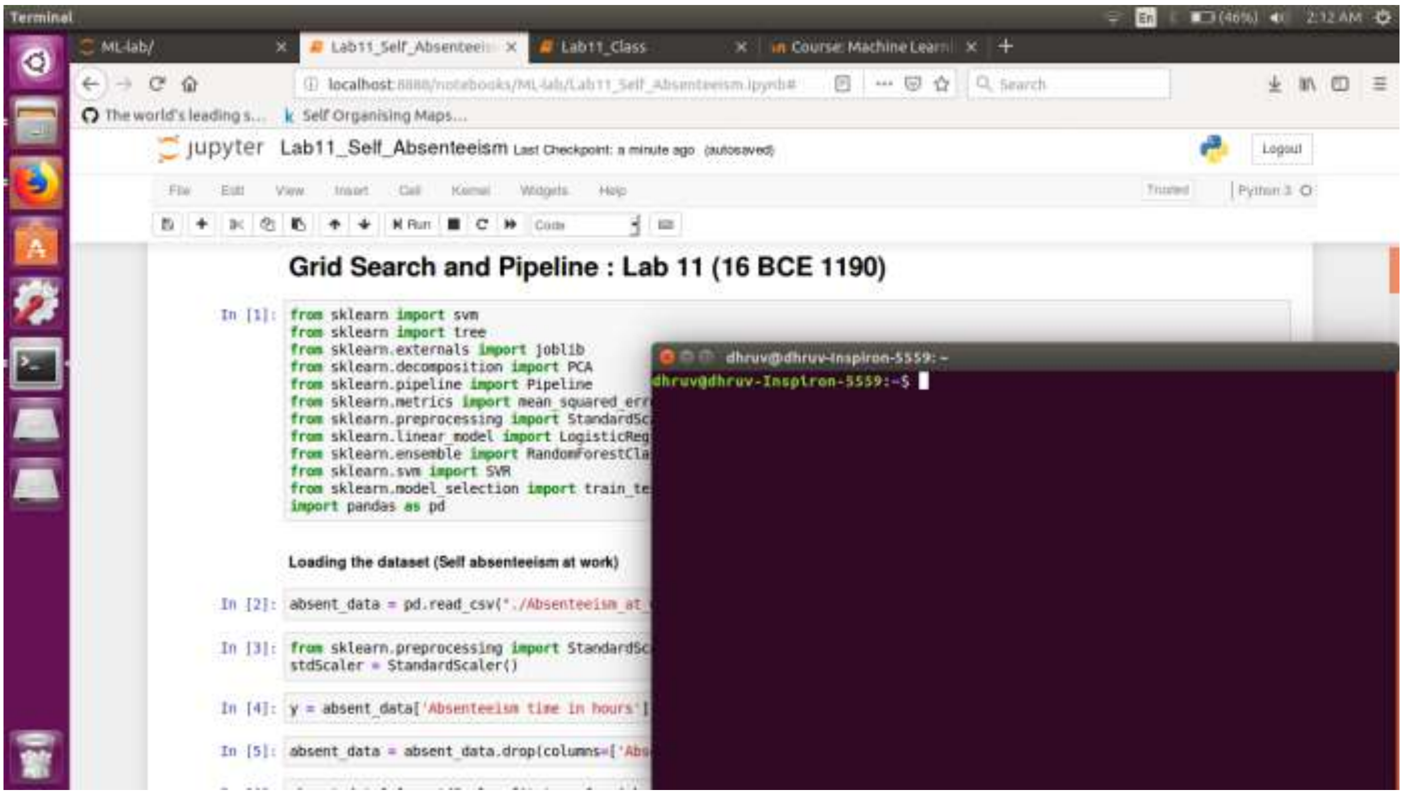
Constructing pipelines for data processin

```
sklearn.pipeline.Pipeline(steps, memory=None
```

```
In [3]: pipe_lr = Pipeline([('scl', StandardScaler()
                            ('pca', PCA(n_components
                            ('clf', LogisticRegressi
```

```
dhruv@dhruv-Inspiron-5559:~$
```

Chosen dataset – Absenteeism at work


Terminal
ML-lab/
Lab11_Self_Absenteeis
Lab11_Class
Course: Machine Learni
localhost:8888/notebooks/ML-lab/Lab11_Self_Absenteeism.ipynb#
Search
The world's leading s...
Self Organising Maps...
jupyter Lab11_Self_Absenteeism Last Checkpoint: a minute ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
dhruv@dhruv-Inspiron-5559: ~

Grid Search and Pipeline : Lab 11 (16 BCE 1190)

```
In [1]: from sklearn import svm
        from sklearn import tree
        from sklearn.externals import joblib
        from sklearn.decomposition import PCA
        from sklearn.pipeline import Pipeline
        from sklearn.metrics import mean_squared_err
        from sklearn.preprocessing import StandardSc
        from sklearn.linear_model import LogisticReg
        from sklearn.ensemble import RandomForestCla
        from sklearn.svm import SVR
        from sklearn.model_selection import train_te
        import pandas as pd
```

Loading the dataset (Self absenteeism at work)

```
In [2]: absent_data = pd.read_csv("./Absenteeism_at_

In [3]: from sklearn.preprocessing import StandardSc
        stdScaler = StandardScaler()

In [4]: y = absent_data['Absenteeism time in hours']

In [5]: absent_data = absent_data.drop(columns=['Abs
```

```
dhruv@dhruv-Inspiron-5559:~$
```