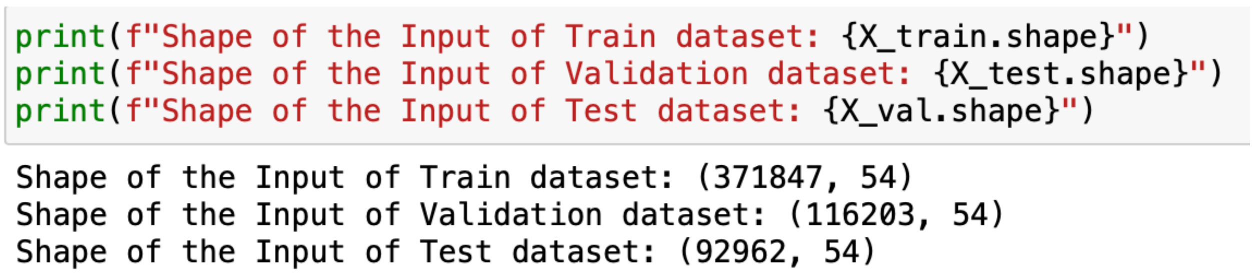
Machine Learning
Fall 2024
Building an ensemble of different models

Khalid Mammadov

Task 1: Votting Classifier:

Part 1: Loading data

In this part, I downloaded the data with pandas and copied the data so that I can work on copied data instead of original onw. The reason behind it because in case of performing something wring on data, I can just go back and copy the original one instead of loading it again and again. Data consists of 581012 rows and 55 columns. After loading this data, I splitted it into three parts: Train →371847, Test → 92962, and Validation →116203: Here is how dataset's shapes are:



```
print(f"Shape of the Input of Train dataset: {X_train.shape}")
print(f"Shape of the Input of Validation dataset: {X_test.shape}")
print(f"Shape of the Input of Test dataset: {X_val.shape}")
```

```
Shape of the Input of Train dataset: (371847, 54)
Shape of the Input of Validation dataset: (116203, 54)
Shape of the Input of Test dataset: (92962, 54)
```

After splitting the data, I scaled my features with standart scaler to reduce computation cost.

Part 2: Modeling

In this part, I train 5 different models: RandomForestClassifier, ExtraTreesClassifier, SGDClassifier, LinearSVC and MLPClassifier. I train my models with scaled data and evaluate each of them on new unseen data which is validation dataset. Results for each dataset are:

- RandomForest: 0.95
- ExtraTrees: 0.95
- SGD: 0.71
- LinearSVC: 0.71
- MLP: 0.86

The best performance belongs to tree models → Random Forest and Extra Trees. SGD and LinearSVC performed poor as they are linear models which is not the best for our dataset.

Part 3: Ensembling

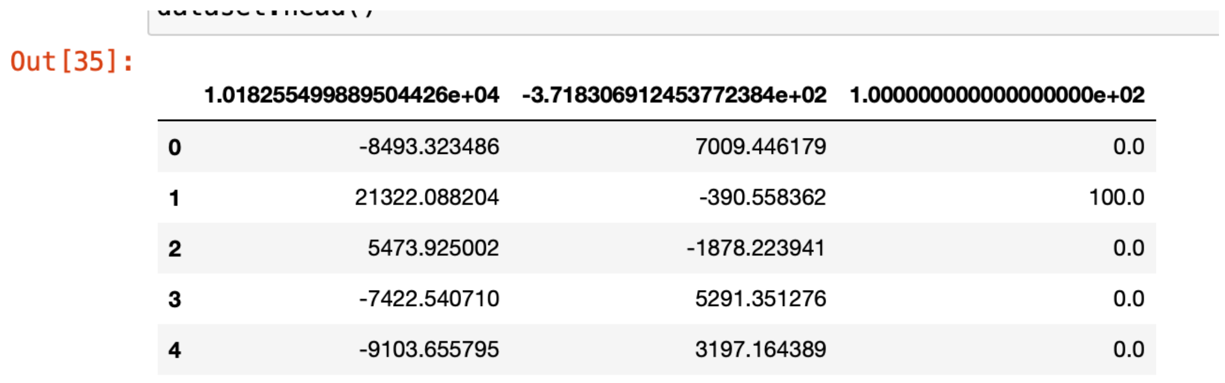
After training every individual model, I trained two Voting classifiers: hard and soft. For hard voting classifier, I passed all the models I trained previously as estimators. The reason is that all modles have predict attribute and hard voting classifier satisfies with it. I trained it on scaled train data and evaluated on vlaidation data. Accuracy score for this classifier is 90% which is better than individual SGDClassifier, LinearSVC and MLPClassifier whose accuracies were less than 90%. To check if there is any model that affects bad to my classifier, I started to remove one of them each time and evaluated hard voting classifier on validation data without that specific model to see if its absence affect good or bad to my voting classifier. As a result, I realized that if I remove one of the linear classifiers, then accuracy increases from 90% to about 92%. However, removing both of them and keeping only non-linear models increases the accuracy even more → to 95%. Therefore, I keep only three estimators in my final hard voting classifier: RandomForest, ExtraTrees, MLP.

When it comes to soft voting classifier, I only passed a RandomForest, ExtraTrees and and MLP as estimators. The reason is that these models have predict_proba attribute which is necessary for soft voting classifier to calculate the scores. I trained it on scaled train data and evaluated on vlaidation data. Accuracy score for this classifier is 94% which is better than individual MLPClassifier and hard voting classifier. To check if there is any model that affects bad to my classifier, I started to remove one of them each time and evaluated soft voting classifier on validation data without that specific model to see if its absence affect good or bad to my voting classifier. As a result, I realized that only removing MLP affects good by increasing accuracy of my model from 94% to 95%. Therefore, I removed this model. I did not train soft voting classifier on two other models (Random forest and Extra Tree) again as I already did when I removed MLP and calculated the accuracy → 95.15%

Task 2: Random Forest:

Part 1: Loading data

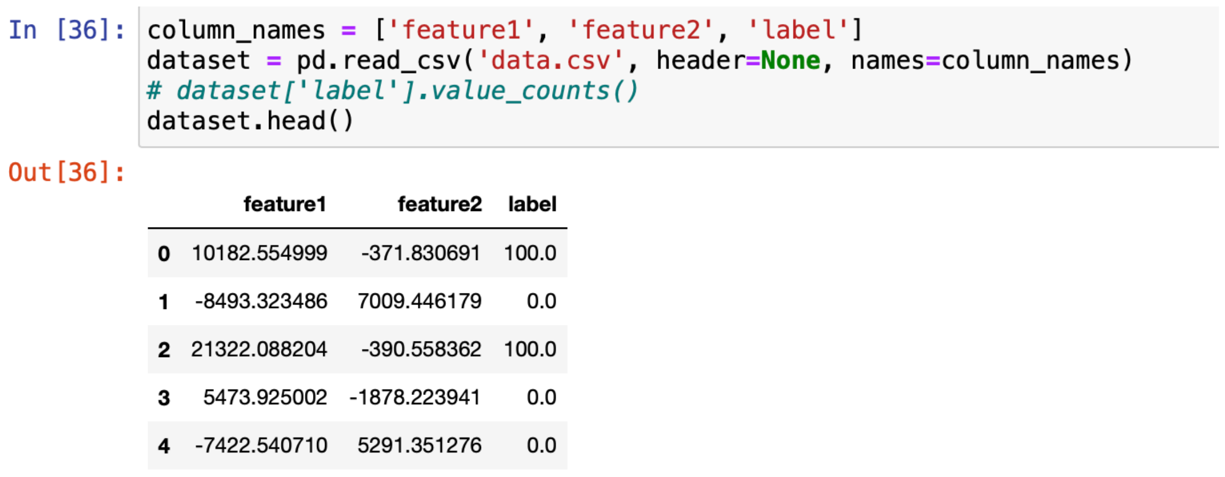
In this part, I downloaded the data with pandas. When I forst downloaded it, the column names were too long and complex. Here is how it looked:



Out[35]:

	1.018255499889504426e+04	-3.718306912453772384e+02	1.000000000000000000e+02
0	-8493.323486	7009.446179	0.0
1	21322.088204	-390.558362	100.0
2	5473.925002	-1878.223941	0.0
3	-7422.540710	5291.351276	0.0
4	-9103.655795	3197.164389	0.0

Therefore, I changed column names and load the dataset again. Here is the code and new dataset look:



In [36]:
```python
column_names = ['feature1', 'feature2', 'label']
dataset = pd.read_csv('data.csv', header=None, names=column_names)
# dataset['label'].value_counts()
dataset.head()
```

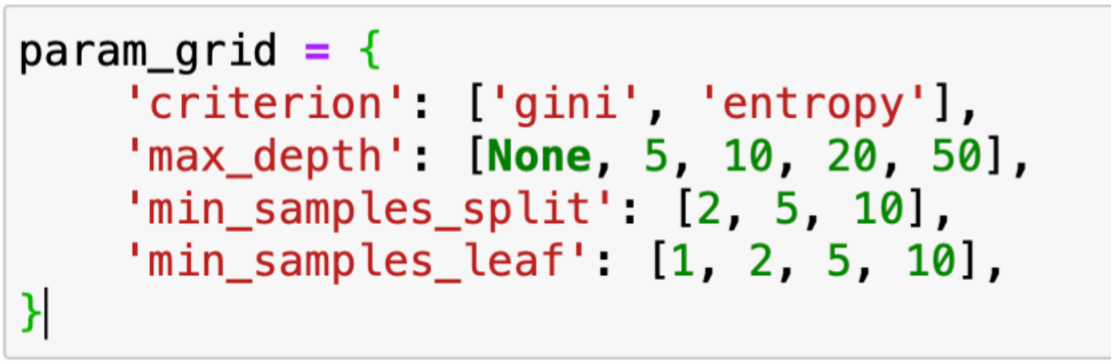
Out[36]:

	feature1	feature2	label
0	10182.554999	-371.830691	100.0
1	-8493.323486	7009.446179	0.0
2	21322.088204	-390.558362	100.0
3	5473.925002	-1878.223941	0.0
4	-7422.540710	5291.351276	0.0

After loading the dataset, I extracted first two columns as X by dropping the last column and I put this column to y. Then, I splited the data into test (20%) and train (80%). Finally, scaled my features with Standart Scaler.

Part 2: Modeling

In this part, I trained Decision Tree Classifier and tune the hyperparameters with GridSearchCv. First of all, I initialized the parameters as param_grid dictionary:

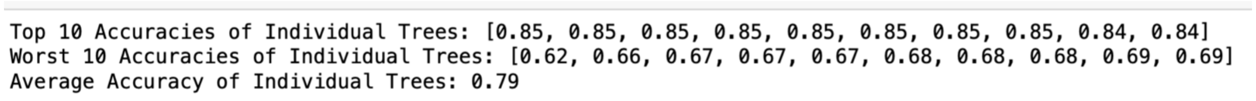


```
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 20, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5, 10],
}
```

Then, found the best hyperparamters with GridSearchCV. The best ones are:

- criterion: entropy,
- max_depth: 5,
- min_samples_leaf: 10
- min_samples_split: 2

With these parameters, accuracy on Test dataset was 86%. I did not need to train best model on the whole dataset since it has already trained. After this, I took 1200 subsets of training dataset each contains 100 samples. I used indices for this. With ShuffleSplit(n_splits=1200, train_size=100, random_state=42) and passing my train dataset, I got 1200 arrays contains 100 random samples from my train dataset. Then, I used these subsets to train the best model I got previously. After training it each time, I evaluated the model on test dataset and I always got less accuracy. The reason is that, since we train the model on just a few (100) numbers of sample, model cannot learn the data well and it cannot generalize well when it sees completely new dataset. Here, I put best and worst accuracies:



```
Top 10 Accuracies of Individual Trees: [0.85, 0.85, 0.85, 0.85, 0.85, 0.85, 0.85, 0.85, 0.84, 0.84]
Worst 10 Accuracies of Individual Trees: [0.62, 0.66, 0.67, 0.67, 0.67, 0.68, 0.68, 0.68, 0.69, 0.69]
Average Accuracy of Individual Trees: 0.79
```

In the final part, I got predictions from all 1200 trees, put them in numpy array. After this, I kept most frequent (mode) prediction for each row. So, my final prediction consisted of these most frequent predictions. I found accuracy of doing so, its accuracy was 83% which is still less than DT. The main reason is that when I use these trees, several of them give wrong prediction as they cannot learn the dataset well because of low number of train dataset. As a result, it affects overall result (if more than half of the tress gave wrong prediction, result is also wrong as mode takes this wrong prediction).