Data Science Lab: Lab 5

Submit:

1. A pdf of your notebook with solutions.
2. A link to your colab notebook or also upload your .ipynb if not working on colab.

Goals of this Lab

1. Random Forests
2. Boosting
3. Playing with Ensembling packages, including XGBoost and CatBoost
4. One more time: Revisiting CIFAR-10 and MNIST
5. Getting ready for Kaggle

We will soon open a Kaggle competition made for this class. In that one, you will be participating on your own. This is an intro to get us started, and also an excuse to work with regularization and regression which we have been discussing. You'll revisit some problems from earlier labs, this time using Random Forests, and Boosting. In particular, you should take this opportunity to become familiar with some very useful packages for boosting. I recommend not only the boosting packages in scikit-learn, but also XGBoost, GBM Light, CatBoost and possibly others. You have to download these and get them running, and then read their documentation to figure out how they work, what the hyperparameters are, etc.

Also, the metric we will use in the Kaggle competition is AUC. We will discuss this. In the meantime, you may want to understand how it works. At least one key thing to remember: to get a good AUC score, you need to submit a soft score (probabilities) and not rounded values (i.e., not 0s and 1s).

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import cupy as cp
```

Problem 1: Revisiting Logistic Regression and MNIST

We have played with the handwriting recognition problem (the MNIST data set) using decision trees. We have also considered the same problem using multi-class Logistic Regression in a previous Lab. We revisit this one more time.

Part 1: Use Random Forests to try to get the best possible *test accuracy* on MNIST. This involves getting acquainted with how Random Forests work, understanding their parameters, and therefore using Cross Validation to find the best settings. How well can you do? You should use the accuracy metric, since this is what you used in the previous Lab -- therefore this will allow you to compare your results from Random Forests with your results from L1- and L2- Regularized Logistic Regression.

What are the hyperparameters of your best model?

Part 2: Use Boosting to do the same. Take the time to understand how XGBoost works (and/or other boosting packages available -- CatBoost is also another favorite). Try your best to tune your hyper-parameters. As added motivation: typically the winners and near-winners of the Kaggle competition are those that are best able to tune and cross validate XGBoost. What are the hyperparameters of your best model?

```
from sklearn.datasets import fetch_openml

from sklearn.metrics import accuracy_score

from sklearn.model_selection import train_test_split

from sklearn.model_selection import RandomizedSearchCV

X, y = fetch_openml('mnist_784', version=1, return_X_y=True)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/datasets/_openml.py:968: FutureWarning: The default value of `parser` will change from `
  warn(
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)
```

Part 1

```
from sklearn.ensemble import RandomForestClassifier

params = {
    'max_depth': np.arange(10, 110, 10),
    'n_estimators': np.arange(50, 250, 50),
    'min_samples_leaf': np.arange(1, 11, 1),
    'min_samples_split': np.arange(2, 11, 1),
}
print('Parameters:', params)
```

```
Parameters: {'max_depth': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100]), 'n_estimators': array([ 50, 100, 150, 200]), 'min_s
```

```
clf = RandomForestClassifier(n_jobs=-1)

rand_forest_clf = RandomizedSearchCV(clf, params, verbose=3, cv=3, n_jobs=-1, n_iter=50)

rand_forest_clf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
▸ RandomizedSearchCV
▸ best_estimator_: RandomForestClassifier
  ▸ RandomForestClassifier
```

```
y_pred = rand_forest_clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print('Accuracy:', accuracy)

print('Best Hyperparameters:', rand_forest_clf.best_params_)
```

```
Accuracy: 0.9684285714285714
Best Hyperparameters: {'n_estimators': 200, 'min_samples_split': 7, 'min_samples_leaf': 1, 'max_depth': 90}
```

Answer:

Accuracy: 0.9684285714285714

Best Hyperparameters: {'n_estimators': 200, 'min_samples_split': 7, 'min_samples_leaf': 1, 'max_depth': 90}

Part 2

```
from xgboost import XGBClassifier

y_train = [int(i) for i in y_train]

y_test = [int(i) for i in y_test]

params = {
```

```
    'max_depth': [5, 10, 15, 20],

    'n_estimators': [100, 110, 120, 130],

    'grow_policy': ['depthwise', 'lossguide'],

    'booster': ['gbtree', 'gblinear', 'dart']

}

print('Parameters:', params)
```

```
Parameters: {'max_depth': [5, 10, 15, 20], 'n_estimators': [100, 110, 120, 130], 'grow_policy': ['depthwise', 'lossguide'], 'booster': [
```

```
clf = XGBClassifier(tree_method='hist', device='cuda', n_jobs=-1, verbosity=0)

xgbrf_clf = RandomizedSearchCV(clf, params, cv=3, n_jobs=-1, n_iter=10)

xgbrf_clf.fit(X_train, y_train)


y_pred = xgbrf_clf.predict(cp.array(X_test))

accuracy = accuracy_score(y_test, y_pred)

print('Accuracy:', accuracy)

print('Best Hyperparameters:', xgbrf_clf.best_params_)
```

```
Accuracy: 0.9764285714285714
Best Hyperparameters: {'n_estimators': 100, 'max_depth': 5, 'grow_policy': 'depthwise', 'booster': 'dart'}
```

Answer:

Accuracy: 0.9764285714285714

Best Hyperparameters: {'n_estimators': 100, 'max_depth': 5, 'grow_policy': 'depthwise', 'booster': 'dart'}

Problem 2: Revisiting Logistic Regression and CIFAR-10

Now that you have your pipeline set up, it should be easy to apply the above procedure to CIFAR-10. If you did something that takes significant computation time, keep in mind that CIFAR-10 is a few times larger.

Part 1: What is the best accuracy you can get on the test data, by tuning Random Forests? What are the hyperparameters of your best model?

Part 2: What is the best accuracy you can get on the test data, by tuning XGBoost? What are the hyperparameters of your best model?

```
from sklearn.datasets import fetch_openml

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.model_selection import RandomizedSearchCV


X, y = fetch_openml('CIFAR_10_small', version=1, return_X_y=True)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/datasets/_openml.py:968: FutureWarning: The default value of `parser` will change from `
  warn(
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)
```

Part 1

```
from sklearn.ensemble import RandomForestClassifier


params = {
```

```
    'max_depth': np.arange(10, 110, 10),

    'n_estimators': np.arange(50, 250, 50),

    'min_samples_leaf': np.arange(1, 11, 1),

    'min_samples_split': np.arange(2, 11, 1),

}

print('Parameters:', params)
```

```
Parameters: {'max_depth': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100]), 'n_estimators': array([ 50, 100, 150, 200]), 'min_s
```

```
clf = RandomForestClassifier(n_jobs=-1)

rand_forest_clf = RandomizedSearchCV(clf, params, verbose=3, cv=3, n_jobs=-1, n_iter=25)

rand_forest_clf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 25 candidates, totalling 75 fits
```

RandomizedSearchCV

best_estimator_: RandomForestClassifier

RandomForestClassifier

```
y_pred = rand_forest_clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print('Accuracy:', accuracy)

print('Best Hyperparameters:', rand_forest_clf.best_params_)
```

```
Accuracy: 0.45275
Best Hyperparameters: {'n_estimators': 200, 'min_samples_split': 8, 'min_samples_leaf': 3, 'max_depth': 90}
```

Answer:

Accuracy: 0.45275

Best Hyperparameters: {'n_estimators': 200, 'min_samples_split': 8, 'min_samples_leaf': 3, 'max_depth': 90}

Part 2

```
from xgboost import XGBClassifier

y_train = [int(i) for i in y_train]

y_test = [int(i) for i in y_test]

params = {

    'max_depth': [4, 6, 8, 10],

    'n_estimators': [100, 105, 110],

    'grow_policy': ['depthwise', 'lossguide'],

    'booster': ['gbtree', 'gblinear', 'dart']

}

print('Parameters:', params)
```

```
Parameters: {'max_depth': [4, 6, 8, 10], 'n_estimators': [100, 105, 110], 'grow_policy': ['depthwise', 'lossguide'], 'booster': ['gbtree
```

```
clf = XGBClassifier(tree_method='hist', device='cuda', n_jobs=-1, verbosity=0)

xgbrf_clf = RandomizedSearchCV(clf, params, cv=3, n_jobs=-1, n_iter=5)

xgbrf_clf.fit(X_train, y_train)


y_pred = xgbrf_clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print('Accuracy:', accuracy)

print('Best Hyperparameters:', xgbrf_clf.best_params_)
```

```
Accuracy: 0.507
Best Hyperparameters: {'n_estimators': 100, 'max_depth': 8, 'grow_policy': 'lossguide', 'booster': 'dart'}
```

Answer:

Accuracy: 0.507

Best Hyperparameters: {'n_estimators': 100, 'max_depth': 8, 'grow_policy': 'lossguide', 'booster': 'dart'}

Problem 3: Revisiting Kaggle

This is a continuation of Problem 2 from Lab 3. You already did some first steps there, including making a Kaggle account, and trying ridge and lasso linear regression. You also tried stacking.

Part 1 (Nothing to hand in) Revisit Lab 3 and your answers there.

Part 2: Train a gradient boosting regression, e.g., using XGBoost. What score can you get just from a single XGB? (you will need to optimize over its parameters).

Part 3: Do your best to get a more accurate model. Try feature engineering and stacking many models. You are allowed to use any public tool in python. No non-python tools allowed. State what hyperparameters and models you tried, and the corresponding train/test error.

Part 4: (Optional) Read the Kaggle forums, tutorials and Kernels in this competition. This is an excellent way to learn. Include in your report if you find something in the forums you like, or if you made your own post or code post, especially if other Kagglers liked or used it afterwards.

Other: Be sure to read and learn the rules of Kaggle! No sharing of code or data outside the Kaggle forums. Every student should have their own individual Kaggle account and teams can be formed in the Kaggle submissions with your Lab partner. This is more important for live competitions of course.

In the real in-class Kaggle competition (which will be next), you will be graded based on your public score (include that in your report) and also on the creativity of your solution. In your report, due after the competition closes, you will explain what worked and what did not work. Many creative things will not work, but you will get partial credit for developing them. You can start thinking about this now.

```
from sklearn.model_selection import RandomizedSearchCV
import matplotlib
from scipy.stats import skew
from scipy.stats.stats import pearsonr
```

```
/tmp/ipykernel_30/4078800519.py:4: DeprecationWarning: Please import `pearsonr` from the `scipy.stats` namespace; the `scipy.stats.stats
  from scipy.stats.stats import pearsonr
```

```
from google.colab import files


uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
train = pd.read_csv('/kaggle/input/lab-5-data/train.csv')

test = pd.read_csv('/kaggle/input/lab-5-data/test.csv')

# Preprocessing from lab 3
all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],test.loc[:,'MSSubClass':'SaleCondition']))
matplotlib.rcParams['figure.figsize'] = (12.0, 6.0)
```

```
prices = pd.DataFrame({"price":train["SalePrice"], "log(price + 1)":np.log1p(train["SalePrice"])})
#log transform the target:
train["SalePrice"] = np.log1p(train["SalePrice"])
#log transform skewed numeric features:
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index
skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna())) #compute skewness
skewed_feats = skewed_feats[skewed_feats > 0.75]
skewed_feats = skewed_feats.index
all_data[skewed_feats] = np.log1p(all_data[skewed_feats])
all_data = pd.get_dummies(all_data)
#filling NA's with the mean of the column:
all_data = all_data.fillna(all_data.mean())
#creating matrices for sklearn:
X_train = all_data[:train.shape[0]]
X_test = all_data[train.shape[0]:]
y_train = train.SalePrice
# Finished Preprocessing
```

Part 2

```
from xgboost import XGBRegressor
```

```
params = {

    'max_depth': [4, 6, 8, 10, 12],

    'n_estimators': [100, 125, 150],

    'grow_policy': ['depthwise', 'lossguide'],

    'booster': ['gbtree', 'gblinear', 'dart']

}

print('Parameters:', params)
```

```
Parameters: {'max_depth': [4, 6, 8, 10, 12], 'n_estimators': [100, 125, 150], 'grow_policy': ['depthwise', 'lossguide'], 'booster': ['gb
```

```
reg = XGBRegressor(tree_method='hist', device='cuda', n_jobs=-1, verbosity=0)

xgb_reg = RandomizedSearchCV(reg, params, cv=3, n_jobs=-1, n_iter=15)

xgb_reg.fit(X_train, y_train)
```

```
RandomizedSearchCV
estimator: XGBRegressor
XGBRegressor
```

```
# score using cross validation

print('Score:', xgb_reg.best_score_)

print('Best hyperparameters:', xgb_reg.best_params_)
```

```
Score: 0.8806253675978842
Best hyperparameters: {'n_estimators': 100, 'max_depth': 4, 'grow_policy': 'lossguide', 'booster': 'gbtree'}
```

Answer:

Score: 0.8806253675978842

Best hyperparameters: {'n_estimators': 100, 'max_depth': 4, 'grow_policy': 'lossguide', 'booster': 'gbtree'}

Part 3

```
!pip install catboost
```

Show hidden output

```
from sklearn.linear_model import Ridge, Lasso

from catboost import CatBoostRegressor

# using best alphas from lab 3

ridge = Ridge(alpha=10)

lasso = Lasso(alpha=0.0005)


ridge.fit(X_train, y_train)

lasso.fit(X_train, y_train)
```

```
Lasso
Lasso(alpha=0.0005)
```

```
params = {

    'depth': [4, 6, 8, 10],

    'learning_rate': [0.01, 0.05, 0.1]

}

print('Parameters:', params)


cat = CatBoostRegressor(silent=True, task_type='CPU')

cat_reg = RandomizedSearchCV(cat, params, cv=3, verbose=3, n_jobs=-1, n_iter=10)

cat_reg.fit(X_train, y_train)
```

```
Parameters: {'depth': [4, 6, 8, 10], 'learning_rate': [0.01, 0.05, 0.1]}
Fitting 3 folds for each of 10 candidates, totalling 30 fits
RandomizedSearchCV
estimator: CatBoostRegressor
CatBoostRegressor
```

```
ridge_pred = ridge.predict(X_train)

lasso_pred = lasso.predict(X_train)

cat_reg_pred = cat_reg.predict(X_train)


X_train_new = X_train.copy()

# stacking

X_train_new['ridge_pred'] = ridge_pred

X_train_new['lasso_pred'] = lasso_pred

X_train_new['cat_reg_pred'] = cat_reg_pred

X_train_new.head()
```

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	SaleTyp
0	4.110874	4.189655	9.042040	7	5	2003	2003	5.283204	6.561031	0.0	...	
1	3.044522	4.394449	9.169623	6	8	1976	1976	0.000000	6.886532	0.0	...	
2	4.110874	4.234107	9.328212	7	5	2001	2002	5.093750	6.188264	0.0	...	
3	4.262680	4.110874	9.164401	7	5	1915	1970	0.000000	5.379897	0.0	...	
4	4.110874	4.442651	9.565284	8	5	2000	2000	5.860786	6.486161	0.0	...	

5 rows × 290 columns

```
params = {

    'max_depth': [4, 6, 8, 10, 12],

    'n_estimators': [100, 125, 150],

    'grow_policy': ['depthwise', 'lossguide'],

    'booster': ['gbtree', 'gblinear', 'dart']

}

print('Parameters:', params)
```

```
Parameters: {'max_depth': [4, 6, 8, 10, 12], 'n_estimators': [100, 125, 150], 'grow_policy': ['depthwise', 'lossguide'], 'booster': ['gb
```

```
from xgboost import XGBRegressor

reg = XGBRegressor(tree_method='hist', device='cuda', n_jobs=-1, verbosity=0)

xgbrf_reg = RandomizedSearchCV(reg, params, cv=3, verbose=3, n_jobs=-1, n_iter=15)

xgbrf_reg.fit(X_train_new, y_train)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
```

▸ **RandomizedSearchCV**
▸ **estimator: XGBRegressor**
▸ XGBRegressor

```
print('Score:', xgbrf_reg.best_score_)

print('Best hyperparameters:', xgbrf_reg.best_params_)
```

```
Score: 0.9737366507559516
Best hyperparameters: {'n_estimators': 100, 'max_depth': 4, 'grow_policy': 'depthwise', 'booster': 'gbtree'}
```

```
from sklearn.metrics import mean_squared_error

best = xgbrf_reg.best_estimator_

y_train_pred = best.predict(X_train_new)

train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))

print('Train Error (RMSE):', train_rmse)


ridge_pred = ridge.predict(X_test)
lasso_pred = lasso.predict(X_test)
cat_reg_pred = cat_reg.predict(X_test)


X_test_new = X_test.copy()

# stacking

X_test_new['ridge_pred'] = ridge_pred
```

```
X_test_new['lasso_pred'] = lasso_pred

X_test_new['cat_reg_pred'] = cat_reg_pred


y_test_pred = np.expm1(best.predict(X_test_new))
df_submission = pd.DataFrame({'Id':test.Id, 'SalePrice':y_test_pred})
df_submission.to_csv('submission.csv', index=False)
df_submission.head()
```

	Id	SalePrice
0	1461	127406.031250
1	1462	156469.953125
2	1463	193869.890625
3	1464	196019.984375
4	1465	180442.375000

Answer:

Train RMSE: 0.1204000050364664

Test RMSE (submission to Kaggle): 0.13088