Assignment 02 – Random Forest and Stacking

Project Group 7 - Tianqi Zhou - tz232

Initialization

In [ ]:
```
# import libraries required
import random
import pandas as pd

# set a random seed for the project using the last four digits of uid
random.seed(9334)
```

Load the dataset.

In [ ]:
```
# read the dataset
credit_g07 = pd.read_csv('creditcard.csv',header=0)
```

Show first 6 data points using head().

In [ ]:
```
# show the first records using .head()
credit_g07.head(6)
```

Out[ ]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.3
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.5
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.5

6 rows × 31 columns

Describe pandas Dataframe by using describe.

In [ ]:
```
# use .describe() to get the stastics of the dataset
credit_g07.describe()
```

Out[ ]:

	Time	V1	V2	V3	V4	V5
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01

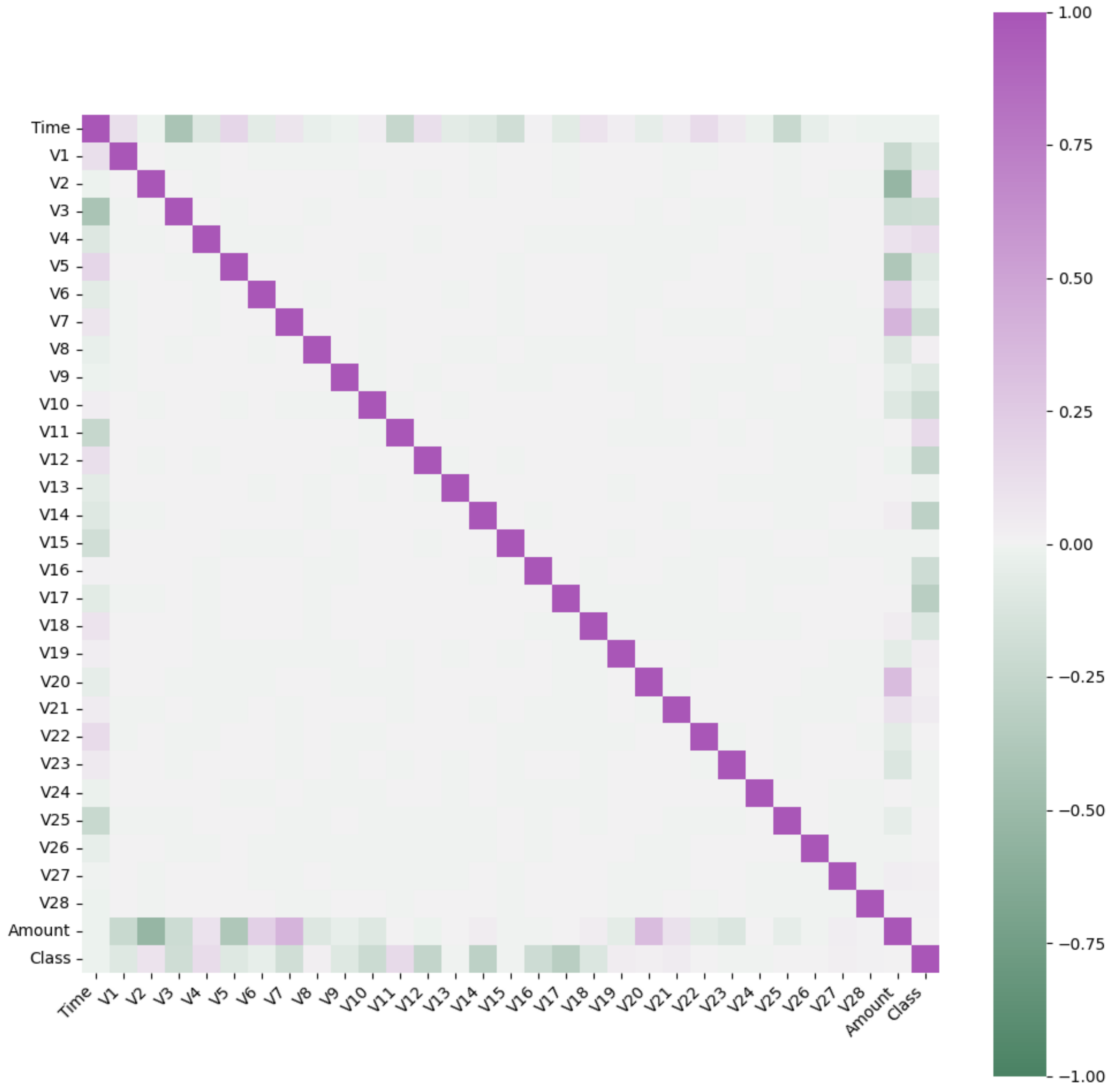
8 rows × 31 columns

Show correlation heat plot of the entire dataset using matplotlib and sns, choose any color pallet (except blue) you like (experiment).

In [ ]:

```python
from matplotlib import rcParams
import seaborn as sns

# change the default setting of the figure using rcParams
rcParams["figure.figsize"] = 12, 12
rcParams["figure.dpi"] = 100

# calculate the correlation of the variables
corr = credit_g07.corr()

# create the heatmap
ax = sns.heatmap(
    corr,
    vmin=-1,
    vmax=1,
    center=0,
    cmap=sns.diverging_palette(145, 300, s=60, as_cmap=True),
    square=True,
)
# add tick labels
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment="right")
```

```
Out[ ]: [Text(0.5, 0, 'Time'),
         Text(1.5, 0, 'V1'),
         Text(2.5, 0, 'V2'),
         Text(3.5, 0, 'V3'),
         Text(4.5, 0, 'V4'),
         Text(5.5, 0, 'V5'),
         Text(6.5, 0, 'V6'),
         Text(7.5, 0, 'V7'),
         Text(8.5, 0, 'V8'),
         Text(9.5, 0, 'V9'),
         Text(10.5, 0, 'V10'),
         Text(11.5, 0, 'V11'),
         Text(12.5, 0, 'V12'),
         Text(13.5, 0, 'V13'),
         Text(14.5, 0, 'V14'),
         Text(15.5, 0, 'V15'),
         Text(16.5, 0, 'V16'),
         Text(17.5, 0, 'V17'),
         Text(18.5, 0, 'V18'),
         Text(19.5, 0, 'V19'),
         Text(20.5, 0, 'V20'),
         Text(21.5, 0, 'V21'),
         Text(22.5, 0, 'V22'),
         Text(23.5, 0, 'V23'),
         Text(24.5, 0, 'V24'),
         Text(25.5, 0, 'V25'),
         Text(26.5, 0, 'V26'),
         Text(27.5, 0, 'V27'),
         Text(28.5, 0, 'V28'),
         Text(29.5, 0, 'Amount'),
         Text(30.5, 0, 'Class')]
```



Show the Scatterplot matrix for the dataframe (avoid matplotlib and sns for this assignment). You can use Scatterplot Matrix Plotly. Use the code for the second image that shows different colors for classes. In this case, you will get two colors since we have two classes. Also, experiment with visual aspects of the image (not a lot, but an excellent visual will always leave a better impression. you can change color, thickness, font, font size, font color, etc.). No need to explain the plots but do save them in a pdf/svg/png with either static export function or html export function from plotly Interactive HTML Export Plotly.

In [ ]:
```python
import plotly.express as px
import pandas as pd

# define the color labels
colors = {0: '0', 1: '1'}
```

```
# Create a scatterplot matrix
fig = px.scatter_matrix(credit_g07,
                        dimensions=credit_g07.columns[:-1],  # Exclude the 'Class'
                        color=credit_g07['Class'].map(colors),
                        color_discrete_sequence=["#93C572", "purple"],  # Set the c
                        title="Scatterplot Matrix with Color Palette",
                        labels={col: col for col in credit_g07.columns[:-1]})

# update the layout to hide diagonal plots and make the marker smaller
fig.update_traces(diagonal_visible=False,marker=dict(size=3))

# change the figure size to make it clearer
fig.update_layout(
    width=2000,
    height=2000,
)
# save the figure as an interactive html
fig.write_html("scatter_matrix.html")
```

Split the dataset into the Training set and Test set. Choose your preferred split and justify the rationale.

In [ ]:
```
from sklearn.model_selection import train_test_split

# remove the time variable that is irrelevant to our problem
credit_g07 = credit_g07.iloc[:, 1:]

# initialize the independent variable X and dependent variable Y
X_credit_g07 = credit_g07.loc[:, credit_g07.columns != "Class"].to_numpy()
y_credit_g07 = credit_g07.iloc[:, -1:].to_numpy()

# split the dataset into training and testing set (60/40)
X_train_g07, X_test_g07, y_train_g07, y_test_g07 = train_test_split(
    X_credit_g07, y_credit_g07, test_size=0.40, random_state=0
)
```

Answer: Here I used a 60-40 training-test split because in this way, we could get sufficient training data for the model to learn the pattern, and we also have adequate testing data to evaluate the model performance.

Perform classification routine by using LogisticRegression(), KNeighborsClassifier(), DecisionTreeClassifier(), SVC(), GaussianNB(), RandomForestClassifier(), BaggingClassifier(), GradientBoostingClassifier(), XGboostclassifier. Output the accuracy box plot as we have seen in the class (make sure to change regressmod df to classmod. And use an appropriate metric for classification evaluation, for example, accuracy, precision,recall etc). Remember to use the object oriented approach and develop a function (def...), this will be very helpful for the next assignment.

In [ ]:
```python
from sklearn.model_selection import cross_val_score, RepeatedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import GradientBoostingClassifier, BaggingClassifier, Random
from xgboost import XGBClassifier

# define a function for base models we're going to train
def base_models():
  models = dict()
  models["lg"] = LogisticRegression()
  models["KNN"] = KNeighborsClassifier(n_neighbors=3)
  models["Tree"] = DecisionTreeClassifier()
  models["svc"] = SVC()
  models["NB"] = GaussianNB()
  models["Random Forest"] = RandomForestClassifier()
  models["Bagging"] = BaggingClassifier()
  models["GBM"] = GradientBoostingClassifier()
  models["XGB"] = XGBClassifier()
  return models


# define a function to evaluate the list of models using cross validation
def eval_models(model):
  # perform ten fold cross validation
  cv = RepeatedKFold(n_splits=5, n_repeats=2, random_state=1)
  # use accuracy to evaluate the performance of our classification models
  scores = cross_val_score(model, X_train_g07, y_train_g07, scoring='accuracy', cv=
                           error_score='raise')
  return scores
```

In [ ]:
```python
import numpy as np
models_g07 = base_models()

# set lists for the results and model names
results_g07, names_g07 = list(), list()

for name, model in models_g07.items():
  scores = eval_models(model)
  results_g07.append(scores)
  names_g07.append(name)
  print('>%s %.3f (%.3f)' % (name, scores.mean(), scores.std()))

# specify our classifier dataframe and the name of the classifiers
classifier_g07 = pd.DataFrame(np.transpose(results_g07), columns = ["lg","KNN","Tre
classifier_g07 = pd.melt(classifier_g07.reset_index(), id_vars='index',value_vars=

# plot the result in box plot
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objs as go

fig = px.box(classifier_g07, x="variable", y="value",color="variable",points='all'
labels={"variable": "Machine Learning Model",
        "value": " Accuracy"
        },title="Model Performance")

# save the figure as an interactive html
from google.colab import drive
```

```
drive.mount('/content/drive')

# Write a sample file to Google Drive
fig.write_html("/content/drive/My Drive/assignment_output/boxplot_for_all_models.ht
```

```
>lg 0.999 (0.000)
>KNN 0.999 (0.000)
>Tree 0.999 (0.000)
>svc 0.999 (0.000)
>NB 0.978 (0.001)
>Random Forest 1.000 (0.000)
>Bagging 0.999 (0.000)
>GBM 0.999 (0.000)
>XGB 1.000 (0.000)
Drive already mounted at /content/drive; to attempt to forcibly remount, call driv
e.mount("/content/drive", force_remount=True).
```

Select the best classifier for level o classifier. Use logistic regression as a second level classifier. Similar to 5 generate the box plot and show the accuracy of each algorithm as well as stacked classifier. Also show the metrices of the above algorithms .

In [ ]:

```
# the best model is XGBoosting classifier
from sklearn.ensemble import StackingClassifier
def get_stacking():
        # define the base models
  level0 = list()
  level0.append(('XGB', XGBClassifier()))

        # define second level model
  level1 = LogisticRegression()

        # define the stacking ensemble
  model = StackingClassifier(estimators=level0, final_estimator=level1, cv=5)
  return model


# define a function for base models we're going to train
def base_models():
  models = dict()
  models["lg"] = LogisticRegression()
  models["XGB"] = XGBClassifier()
  models["Stacked Model"] = get_stacking()
  return models


# define a function to evaluate the list of models using cross validation
def eval_models(model):
  # perform ten fold cross validation
  cv = RepeatedKFold(n_splits=5, n_repeats=2, random_state=1)
  # use accuracy to evaluate the performance of our classification models
  scores = cross_val_score(model, X_train_g07, y_train_g07, scoring='accuracy', cv=
                           error_score='raise')
  return scores

models_g07 = base_models()

# set lists for the results and model names
```

```
results_g07, names_g07 = list(), list()

for name, model in models_g07.items():
  scores = eval_models(model)
  results_g07.append(scores)
  names_g07.append(name)
  print('>%s %.3f (%.3f)' % (name, scores.mean(), scores.std()))


classifier_g07 = pd.DataFrame(np.transpose(results_g07), columns = ["lg", "XGB","st
classifier_g07 = pd.melt(classifier_g07.reset_index(), id_vars='index',value_vars=[


import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objs as go

fig = px.box(classifier_g07, x="variable", y="value",color="variable",points='all',
labels={"variable": "Machine Learning Model",
        "value": " Accuracy"
        },title="Model Performance")

# save the figure as an interactive html
from google.colab import drive

drive.mount('/content/drive')

# Write a sample file to Google Drive
fig.write_html("/content/drive/My Drive/assignment_output/boxplot_for_stack_models.
```

```
>lg 0.999 (0.000)
>XGB 1.000 (0.000)
>Stacked Model 1.000 (0.000)
Drive already mounted at /content/drive; to attempt to forcibly remount, call driv
e.mount("/content/drive", force_remount=True).
```

Export the Pickle model and import it back. Use the imported model to predict the y_test from x_test and report the confusion matrix

In [ ]:

```
import pickle
# reshape the y array to avoid warning
y_train_g07 = y_train_g07.reshape(len(y_train_g07),)
y_test_g07 = y_test_g07.reshape(len(y_test_g07),)

level0 = list()
level0.append(('XGB', XGBClassifier()))

# define second level model
level1 = LogisticRegression()

# define the stacking ensemble
model = StackingClassifier(estimators=level0, final_estimator=level1, cv=5)
model.fit(X_train_g07, y_train_g07)

# save to file in the current working directory
pkl_filename_g07 = "AssignmentPickle.pkl"
with open(pkl_filename_g07, 'wb') as file:
    pickle.dump(model, file)

# load from file
with open(pkl_filename_g07, 'rb') as file:
    pickle_model = pickle.load(file)

# calculate the score and the prediction
score = pickle_model.score(X_test_g07, y_test_g07)
print("Test score: {0:.2f} %".format(100 * score))
y_pred_g07 = pickle_model.predict(X_test_g07)

# report the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test_g07, y_pred_g07)
print(cm)
```

```
Test score: 99.95 %
[[113715      9]
 [    46    153]]
```

Show both text and visual confusion Matrices using scikit learn and matplotlib and explain what the graph tells you and what you did

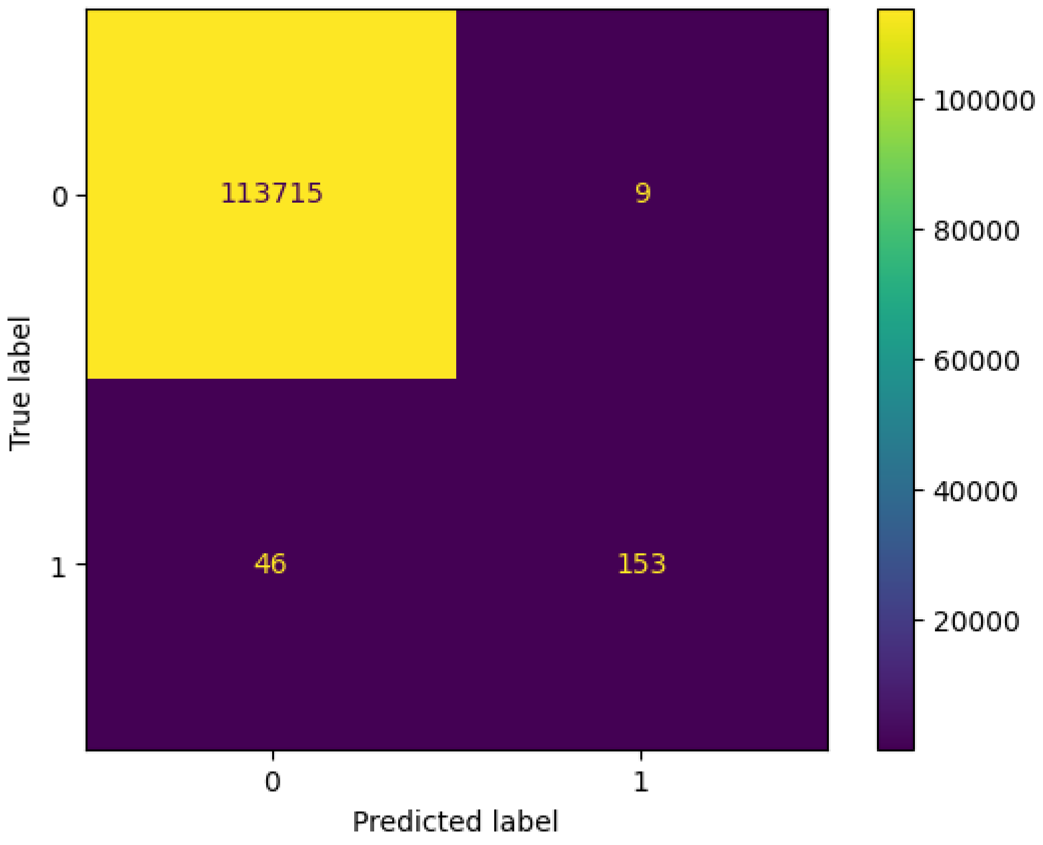
In [ ]:

```
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# set the canvas for the confusion matrix with size = 2.5x2.5 and dpi = 75
plt.figure(figsize=(2.5, 2.5), dpi=75)

# use confusion_matrix from sklearn.metrics to calculate the confusion matrix for t
print(cm)

# use ConfusionMatrixDisplay to visualize the confusion matrix with the model, the
ConfusionMatrixDisplay.from_estimator(model, X_test_g07, y_test_g07)
```

```
[[113715      9]
 [    46    153]]
```

Out[ ]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7d2bc94514e0>

<Figure size 187.5x187.5 with 0 Axes>
```



Answer: The command first sets the canvas for the confusion matrix with size = 2.5x2.5 and dpi = 75. Then it uses confusion_matrix from sklearn.metrics to calculate the confusion matrix for the test set. ConfusionMatrixDisplay command is also used to visualize the confusion matrix with the model, the input matrix, and the true value.

The graph shows that almost all samples (113715 out of 113724) from type 0 are correctly classified. However, almost 1/4 (46 out of 199) of the samples from type 1 are classified wrongly. This might be due to the imbalence nature of our data set with type 0 many times as much as type 1. Therefore, the classifier tends to classify a sample to type 0 instead of type 1.