1 Project Description

=> Implementing your own SVM https://www.kaggle.com/uciml/adult-census-income#adult.csv

2 Dataset preprocessing and interpretation

- ⇨ (part 1) **Abandon the rows that contain ' ?'**

Use,
data = data [(data != '?').all(1)]

To remove the rows with missing values(?)

- ⇨ (Part2) **Find a good representation for them so that they can be used to train a support vector machine.**

Following features are continuous and numerical:

age

fnlwgt

eduction.num

capital.gain

capital.loss

hours.per.week

following are discrete and non-numerical:

workclass

education

marital.status

occupation

relationship

race

sex

native.country

income

In order to model SVM the discrete columns need to be converted which will require label and onehot encoding respectively.

Label encoding: assigns value from 0,1,2,3 by sorting the categories alphabetically.

In order to cope with the errors of miss interpretation we wil use **one hot** encoding of column with more than 2 categories.

workclass = np.unique(data['workclass'])

education = np.unique(data['education'])

martial = np.unique(data['marital.status'])

occupation = np.unique(data['occupation'])

relationship = np.unique(data['relationship'])

race = np.unique(data['race'])

sex = np.unique(data['sex'])

native = np.unique(data['native.country'])

income = np.unique(data['income'])

hours = np.unique(data['hours.per.week']) # Not discrete

Hence we do not consider 'hours.per.week' column as discrete

gle = LabelEncoder()

#transforimng discrete features

workclass_labels = gle.fit_transform(data['workclass'])

education_labels = gle.fit_transform(data['education'])

martial_labels = gle.fit_transform(data['marital.status'])

occupation_labels = gle.fit_transform(data['occupation'])

relationship_labels = gle.fit_transform(data['relationship'])

race_labels = gle.fit_transform(data['race'])

native_labels = gle.fit_transform(data['native.country'])

income_labels = gle.fit_transform(data['income'])

sex_labels = gle.fit_transform(data['sex'])

data_new = data[['age', 'fnlwgt', 'capital.gain', 'capital.loss', 'hours.per.week']].copy()

data_new['workclass'] = workclass_labels

```
data_new['education'] = education_labels

data_new['marital.status'] = martial_labels

data_new['occupation'] = occupation_labels

data_new['relationship'] = relationship_labels

data_new['race'] = race_labels

data_new['native.country'] = native_labels

data_new['income'] = income_labels

data_new['sex'] = sex_labels
```

Need to consider now limitations of label encoder.

```
categorical_subset = pd.get_dummies(data, columns=['workclass', 'education', 'marital.status',
'occupation', 'relationship', 'race', 'native.country'])

categorical_subset.drop('education.num', axis =1)


sex_category = gle.fit_transform(categorical_subset['sex'])

income_category = gle.fit_transform(data['income'])


categorical_subset.drop('sex', axis =1)

categorical_subset.drop('income', axis =1)


categorical_subset['sex'] = sex_category

income_category[income_category==0]=-1

categorical_subset['income'] = income_category
```

⇨ (part3) **Split the dataset for stratified 10-fold-cross validation.**

Using StratifiedKFold from sklearn to implement stratified 10-fold-cross validation.

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=10, random_state=None)
```

```
categorical_subset_2features = categorical_subset[['fnlwgt']].copy()

categorical_subset_2features['reationship'] = relationship_labels

X = categorical_subset_2features.as_matrix()  #StratifiedKfold only accepts numpy matrix

y = categorical_subset['income'].as_matrix()


for train_index, test_index in skf.split(X,y):

   categorical_subset_2features_train_1 = X[train_index]

   categorical_subset_2features_train = 
pd.DataFrame({'fnlwgt':categorical_subset_2features_train_1[:,0],'reationship':categorical_subset_2features_train_1[:,1]})

   y_train_folds_1 = y[train_index]

   y_train_folds = pd.DataFrame({'income':y_train_folds_1[:,]})

   categorical_subset_2features_test_1 = X[test_index]

   categorical_subset_2features_test = 
pd.DataFrame({'fnlwgt':categorical_subset_2features_test_1[:,0],'reationship':categorical_subset_2features_test_1[:,1]})

   y_test_folds_1 = y[test_index]

   y_test_folds = pd.DataFrame({'income':y_test_folds_1[:,]})
```

Here we have data frames for our SVM which comes through the conversion from train/test features was as well as target components.

- ⇨ (Part 4) **Analyze the features and make a scatter plot with the two features that have the highest information gain.**

In order to calculate the information gain of the features, we must have the Final Total entropy of target.

```
def entropy(target_col):

   elements,counts = np.unique(target_col,return_counts = True)

   entropy = np.sum([(-counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts)) for i in 
range(len(elements))])

   return entropy
```

```
total_entropy = entropy(data_new['income'])


def InfoGain(data,split_attribute_name,target_name="income"):
    vals,counts= np.unique(data[split_attribute_name],return_counts=True)


    #Calculate the weighted entropy
    Weighted_Entropy =
np.sum([(counts[i]/np.sum(counts))*entropy(data.where(data[split_attribute_name]==vals[i]).dropna()[
target_name]) for i in range(len(vals))])


    #Calculate the information gain
    Information_Gain = total_entropy - Weighted_Entropy
    return Information_Gain


ig_age = InfoGain(data_new, 'age', 'income')
ig_workclass = InfoGain(data_new, 'workclass', 'income')
ig_fnlwgt = InfoGain(data_new, 'fnlwgt', 'income')
ig_education = InfoGain(data_new, 'education', 'income')
ig_martial = InfoGain(data_new, 'marital.status', 'income')
ig_occupation = InfoGain(data_new, 'occupation', 'income')
ig_relationship = InfoGain(data_new, 'relationship', 'income')
ig_sex = InfoGain(data_new, 'sex', 'income')
ig_race = InfoGain(data_new, 'race', 'income')
ig_gain = InfoGain(data_new, 'capital.gain', 'income')
ig_loss = InfoGain(data_new, 'capital.loss', 'income')
ig_hours = InfoGain(data_new, 'hours.per.week', 'income')
ig_native = InfoGain(data_new, 'native.country', 'income')
```

```
print("Information Gain of age is : ", ig_age)
print("Information Gain of workclass is : ", ig_workclass)
print("Information Gain of fnlwgt is : ", ig_fnlwgt)
print("Information Gain of education is : ", ig_education)
print("Information Gain of marital.status is : ", ig_martial)
print("Information Gain of occupation is : ", ig_occupation)
print("Information Gain of relationship is : ", ig_relationship)
print("Information Gain of sex is : ", ig_sex)
print("Information Gain of race is : ", ig_race)
print("Information Gain of capital.gain is : ", ig_gain)
print("Information Gain of capital.loss is : ", ig_loss)
print("Information Gain of hours.per.week is : ", ig_hours)
print("Information Gain of native.country is : ", ig_native)
```

As per the dataset, we get following information gain.

```
Information Gain of age is :  0.09747880413384746
Information Gain of workclass is :  0.017104479622990443
Information Gain of fnlwgt is :  0.5805932260599063
Information Gain of education is :  0.09339398547736943
Information Gain of marital.status is :  0.15747082217852115
Information Gain of occupation is :  0.09319445792717407
Information Gain of relationship is :  0.16617831761064172
Information Gain of sex is :  0.037406407129768615
Information Gain of race is :  0.008294113320606256
Information Gain of capital.gain is :  0.12094714840490295
Information Gain of capital.loss is :  0.05386983346447127
Information Gain of hours.per.week is :  0.060172937966983975
Information Gain of native.country is :  0.009329014074334951
```

Hence information gain of ‘fnlwgt’ and ‘relationship’ are the top two highest information gain.

Drawing the scatter plot of the two features

#scater plot and analysis

features = list(set(data_new.columns) - set(['income']))

 # Calculate and plot

corr_matrix = data_new[features].corr()

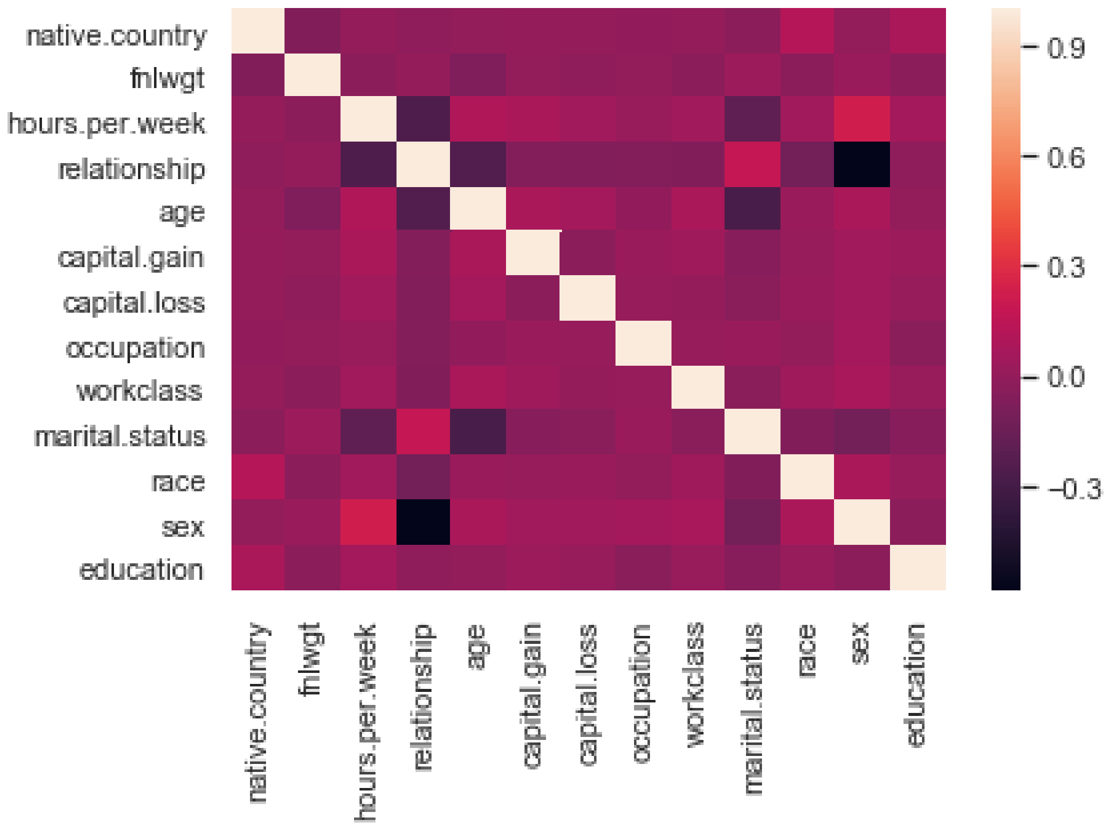
sns.heatmap(corr_matrix)

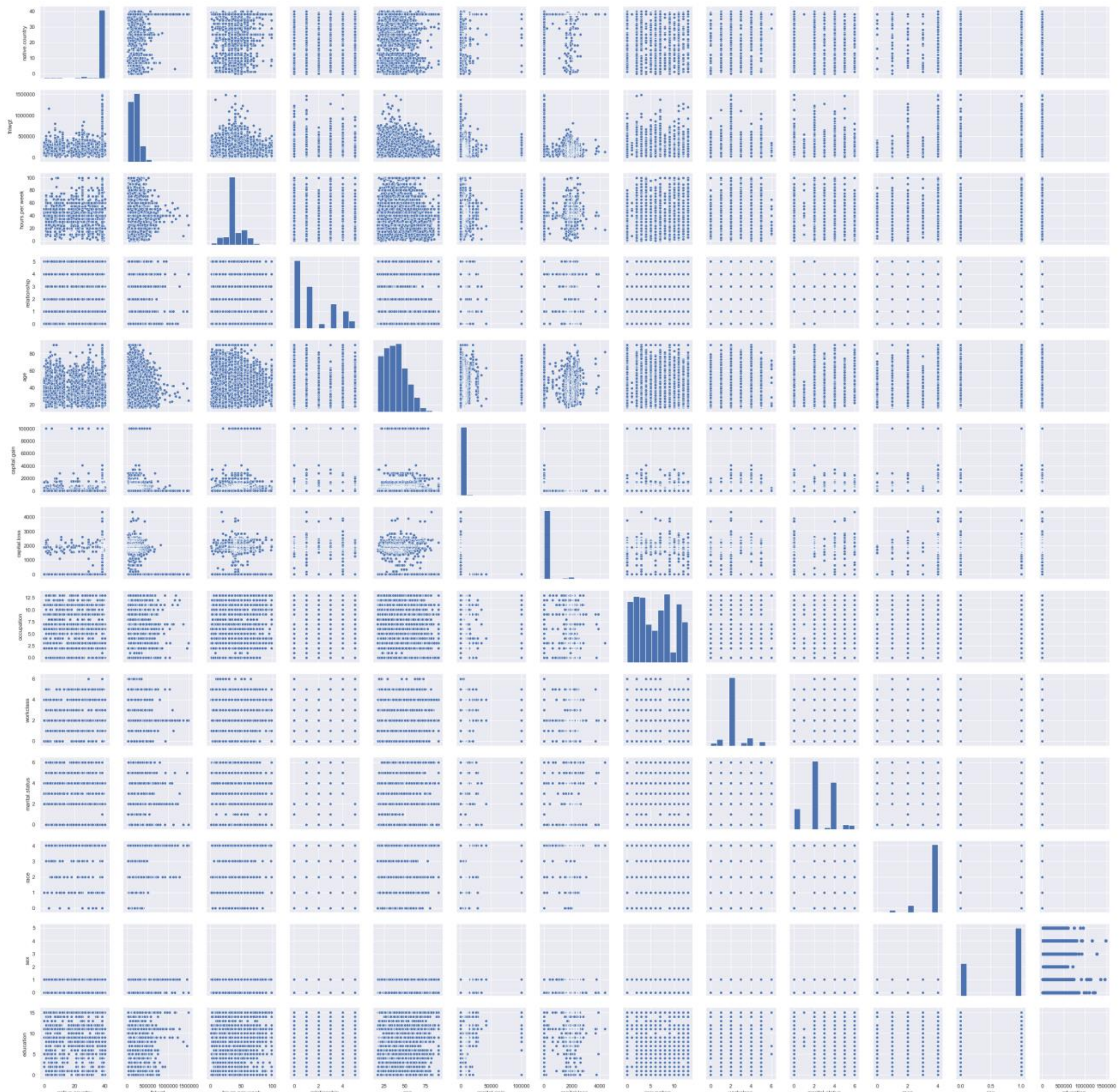
 # pairplot may become very slow with the SVG format

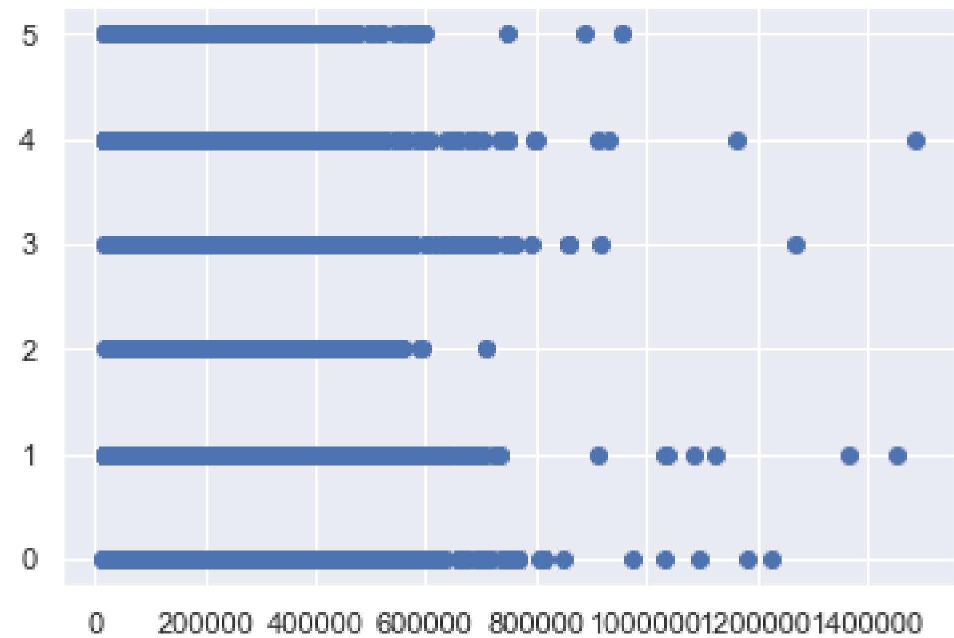
#%config InlineBackend.figure_format = 'png'

sns.pairplot(data_new[features]);

plt.scatter(data_new['fnlwgt'], data_new['relationship'])


native.country
fnlwgt
hours.per.week
relationship
age
capital.gain
capital.loss
occupation
workclass
marital.status
race
sex
education
0.9
0.6
0.3
0.0
−0.3





3.

<u>=>(part1)</u>

Code is present in SVM_3.py

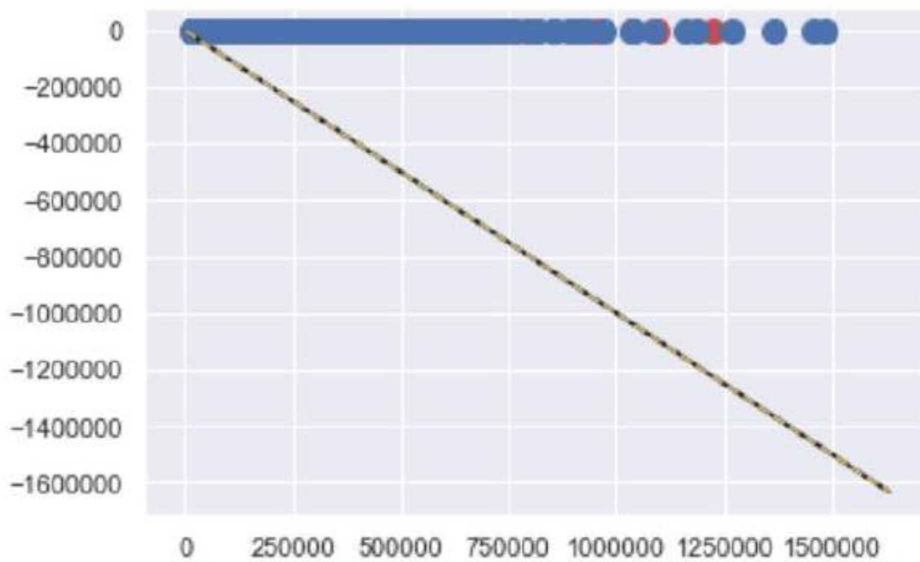
Vector step size of 10%, 0.1%,0.1% are iterated through various values of b. When w passes 0 we can say that we have optimized the algorithm because it follows convex curve.

After training we validate with validate data set and the respective y values.
Then, accuracy would be calculated, after 20 foldes an accuracy of 55.45% was achieved.

=>(part2)

Our svm is based on y_i(x_i*w+b)>=1. Here, the constant b is dependent on slack variable C such that b=C*slack error.
It was observed initially C was as low as 0.423 which gradually increases to 0.5893. Value increases says that points were fitted correctly.

=>(part3)



While analyzing 13 features the accuracy came out to be 0.5789.

Here, the grid search of C is done to optimize the algorithm and the given accuracy is obtained. W can see the value of C increases exponentially.

4.

=> (part 1) **Compare the performance (precision, recall, f1-score, and variance) of different kernels: Linear, RBF, and polynomial.**

Linear:

weights1 = svm(np.array(x_train), y_train, 1000, kernel=linear)

from sklearn.metrics import accuracy_score

predictions1 = predict(x_test, weights1)

print(accuracy_score(y_test, predictions1))

print(f1_score(y_test, predictions1, average='weighted'))

print(recall_score(y_test, predictions1, average='weighted'))

print(explained_variance_score(y_test, predictions1, multioutput='uniform_average'))

Values:

'''

0.7849462365591398

0.6903744008291228

0.7849462365591398

0.7493829464236728

'''

RBF: values

'''

0.7849462365591398

0.6903744008291228

0.7849462365591398

0.7795863423888448

'''

Poly: values

'''

0.6451612903225806

0.6732331937946319

0.6451612903225806

-0.9479452054794524

'''

=> (part 2) **Provide your code and your evaluation method, then explain why the performance is better with your method of choice by using learning curves.**

Following describes the Adaboost method which is used to increase the performance.
We can say SVM is a weak classifier that being known we will use Adaboost to average out the classifier to give a better model using SVM. The problem of poor parameter classification is taken care of here.

import math

```
import numpy as np
import pandas as pd

from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

data=pd.read_csv("adult.csv")

data = data[(data != '?').all(1)]

x = LabelEncoder()

#transforimng discrete features
workclassparam = x.fit_transform(data['workclass'])
educationparam = x.fit_transform(data['education'])
martialparam = x.fit_transform(data['marital.status'])
occupationparam = x.fit_transform(data['occupation'])
relationshipparam = x.fit_transform(data['relationship'])
raceparam = x.fit_transform(data['race'])
nativeparam = x.fit_transform(data['native.country'])
incomeparam = x.fit_transform(data['income'])
sexparam = x.fit_transform(data['sex'])

# dataframe after dealing with discrete features
transformed_data = data[['age', 'fnlwgt', 'capital.gain', 'capital.loss', 'hours.per.week']].copy()

transformed_data['workclass'] = workclassparam
transformed_data['education'] = educationparam
```

```
transformed_data['marital.status'] = martialparam

transformed_data['occupation'] = occupationparam

transformed_data['relationship'] = relationshipparam

transformed_data['race'] = raceparam

transformed_data['native.country'] = nativeparam

transformed_data['income'] = incomeparam

transformed_data['sex'] = sexparam

features = list(set(transformed_data.columns) - set(['income']))

cat_subset = pd.get_dummies(data, columns=['workclass', 'education', 'marital.status', 'occupation', 'relationship', 'race', 'native.country'])

cat_subset.drop('education.num', axis =1)

sex_category = x.fit_transform(cat_subset['sex'])

income_category = x.fit_transform(data['income'])


cat_subset.drop('sex', axis =1)

cat_subset.drop('income', axis =1)

cat_subset['sex'] = sex_category

cat_subset['income'] = income_category

X = cat_subset.drop(['income'],axis=1)

Y = cat_subset['income']

AdaBoost = Ada_Boost_Classifier(n_estimators=400,learning_rate=1,algorithm='SAMME')

AdaBoost.fit(X,Y)

prediction = AdaBoost.score(X,Y)


print('The accuracy is: ',prediction*100,'%')
```

The accuracy comes out to be 0.8536357334548 which is on an average 8% increase from those three different kernel models described in above part.

References:

Support Vector Machine Fundamentals
https://www.youtube.com/watch?v=ZDu3LKv9gOI&list=PLQVvvaa0QuDfKTOs3Keq_kaG2P55YRn5v&index=23