machine learning project

Table of contents

NN output

import pandas as pd ... import numpy as np ... from sklearn.preprocessing import LabelEncoder ... import matplotlib.pyplot as plt ... from sklearn.model*selection import cross_val_score, StratifiedKFold, train_test_split, KFold ... from xgboost import XGBClassifier ... from sklearn.linear_model import LogisticRegression ... from sklearn.ensemble import RandomForestClassifier ... from sklearn.metrics import classification_report ... from sklearn.metrics import roc_curve, roc_auc_score ... import seaborn as sns ... ... # Imports for NN model ... import keras ... from keras.models import Sequential ... from keras.layers import Dense*

---

random state use for initialize status for internal random number

from keras.wrappers.scikit_learn import KerasClassifier

from sklearn.model_selection import GridSearchCV ... ... def build_classifier(optimizer): ... # first step: create a Sequential object, as a sequence of layers. B/C NN is a sequence of layers. ... classifier = Sequential() ... # add the first hidden layer ... classifier.add(Dense(units=6,kernel_initializer='glorot_uniform', ... activation = 'relu')) ... # add the second hidden layer ... classifier.add(Dense(units=6,kernel_initializer='glorot_uniform', ... activation = 'relu')) ... # add the output layer ... classifier.add(Dense(units=1,kernel_initializer='glorot_uniform', ... activation = 'sigmoid')) ... # compiling the NN ... classifier.compile(optimizer=optimizer,loss='binary_crossentropy',metrics=['accuracy']) ... return classifier ... ... classifier = KerasClassifier(build_fn=build_classifier) ... ... # create a dictionary of hyper-parameters to optimize ... parameters = {'batch_size':[10,20], 'nb_epoch': [20,10],'optimizer':['adam']} ... grid_search = GridSearchCV(estimator = classifier, param_grid = parameters, scoring = 'accuracy', cv=10) ... grid_search = grid_search.fit(X_train,y_train) 2345/2345 [==============================] - 5s 2ms/step - loss: 0.5189 - accuracy: 0.7993 WARNING:tensorflow:From C:\Users\nkumr\OneDrive\Desktop\Learning\FTEC 6334 - Financial Applications of Machine Learning\pythonProject\venv\lib\site-packages\tensorflow\python\keras\wrappers\scikit_learn.py:241: Sequential.predict_classes (from tensorflow.python.keras.engine.sequential) is deprecated

and will be removed after 2021-01-01. Instructions for updating: Please use instead: `np.argmax(model.predict(x), axis=-1)` *, if your model does multi-class classification (e.g. if it uses a* `softmax` *last-layer activation).* `(model.predict(x) > 0.5).astype("int32")` *, if your model does binary classification (e.g. if it uses a* `sigmoid` *last-layer activation). 2345/2345 [==============================] - 4s 2ms/step - loss: 0.5019 - accuracy: 0.7897 2345/2345 [==============================] - 4s 2ms/step - loss: 0.4374 - accuracy: 0.7833 2345/2345 [==============================] - 5s 2ms/step - loss: 0.4903 - accuracy: 0.7900 2345/2345 [==============================] - 5s 2ms/step - loss: 0.4439 - accuracy: 0.7927 2345/2345 [==============================] - 4s 2ms/step - loss: 0.5847 - accuracy: 0.7581 2345/2345 [==============================] - 4s 2ms/step - loss: 0.5050 - accuracy: 0.7904 2345/2345 [==============================] - 4s 2ms/step - loss: 0.4319 - accuracy: 0.7832 2345/2345 [==============================] - 5s 2ms/step - loss: 0.4440 - accuracy: 0.7924 2345/2345 [==============================] - 4s 2ms/step - loss: 0.4763 - accuracy: 0.7602 2345/2345 [==============================] - 4s 2ms/step - loss: 0.4926 - accuracy: 0.7567 2345/2345 [==============================] - 4s 2ms/step - loss: 0.4907 - accuracy: 0.7638 2345/2345 [==============================] - 4s 2ms/step - loss: 0.4645 - accuracy: 0.7896 2345/2345 [==============================] - 5s 2ms/step - loss: 0.5063 - accuracy: 0.7910 2345/2345 [==============================] - 5s 2ms/step - loss: 0.4479 - accuracy: 0.7938 2345/2345 [==============================] - 4s 2ms/step - loss: 0.5856 - accuracy: 0.7717 2345/2345 [==============================] - 5s 2ms/step - loss: 0.4963 - accuracy: 0.7976 2345/2345 [==============================] - 4s 2ms/step - loss: 0.4372 - accuracy: 0.7895 2345/2345 [==============================] - 4s 2ms/step - loss: 0.4818 - accuracy: 0.7605 2345/2345 [==============================] - 4s 2ms/step - loss: 0.5131 - accuracy: 0.7679 1173/1173 [==============================] - 4s 3ms/step - loss: 0.5250 - accuracy: 0.7893 1173/1173 [==============================] - 4s 3ms/step - loss: 0.4786 - accuracy: 0.7783 1173/1173 [==============================] - 4s 3ms/step - loss: 0.4888 - accuracy: 0.7920 1173/1173 [==============================] - 4s 3ms/step - loss: 0.6317 - accuracy: 0.7551 1173/1173 [==============================] - 4s 3ms/step - loss: 0.5022 - accuracy: 0.7809 1173/1173 [==============================] - 4s 3ms/step - loss: 0.5273 - accuracy: 0.7903 1173/1173 [==============================] - 4s 3ms/step - loss: 0.4616 - accuracy: 0.7703 1173/1173 [==============================] - 4s 3ms/step - loss: 0.4802 - accuracy:*

0.7621 1173/1173 [==============================] - 4s 3ms/step - loss: 0.4741 - accuracy: 0.7889 1173/1173 [==============================] - 4s 3ms/step - loss: 0.4829 - accuracy: 0.7756 1173/1173 [==============================] - 4s 3ms/step - loss: 0.6009 - accuracy: 0.7600 1173/1173 [==============================] - 4s 3ms/step - loss: 0.5081 - accuracy: 0.7913 1173/1173 [==============================] - 4s 3ms/step - loss: 0.4954 - accuracy: 0.7872 1173/1173 [==============================] - 4s 3ms/step - loss: 0.4866 - accuracy: 0.7767 1173/1173 [==============================] - 4s 3ms/step - loss: 0.4368 - accuracy: 0.7815 1173/1173 [==============================] - 4s 3ms/step - loss: 0.5090 - accuracy: 0.7785 1173/1173 [==============================] - 4s 3ms/step - loss: 0.4816 - accuracy: 0.7572 1173/1173 [==============================] - 4s 3ms/step - loss: 0.4788 - accuracy: 0.7856 1173/1173 [==============================] - 4s 4ms/step - loss: 0.5310 - accuracy: 0.7792 1173/1173 [==============================] - 4s 3ms/step - loss: 0.4622 - accuracy: 0.7892 2605/2605 [==============================] - 5s 2ms/step - loss: 0.4690 - accuracy: 0.7720 ... best_parameters = grid_search.best_params ... best*accuracy = grid_search.best_score* ... ... means = grid*search.cv_results*['mean*test_score'] ... stds = grid_search.cv_results*['std*test_score'] ... params = grid_search.cv_results*['params'] ... for mean, stdev, param in zip(means, stds, params): ... print("%f (%f) with: %r" % (mean, stdev, param)) ... ... print(best_parameters) ... print(best_accuracy) 0.793536 (0.022458) with: {'batch_size': 10, 'nb_epoch': 20, 'optimizer': 'adam'} 0.783437 (0.016503) with: {'batch_size': 10, 'nb_epoch': 10, 'optimizer': 'adam'} 0.787200 (0.013616) with: {'batch_size': 20, 'nb_epoch': 20, 'optimizer': 'adam'} 0.784591 (0.015695) with: {'batch_size': 20, 'nb_epoch': 10, 'optimizer': 'adam'} {'batch_size': 10, 'nb_epoch': 20, 'optimizer': 'adam'} 0.7935355322241583

Requirements for Project

Requirements:

- Define goal for project
- Data Cleaning and Processing
- Modeling Data
- Reporting

Timeline & Milestones

- **11/02:** Data Cleaning and Processing
  - Defining the Problem
  - Original Data
  - Data Cleaning and imputation (using *XGBoost*)
  - Encoding
- **11/16:** Modeling Complete
- **11/29:** Reporting
- **11/30:** Presentation

Code

The Data

Dependent Variables

- **Age**
- **Work Class:** Private, local-gov, never-worked.
- **Fnlwgt:** The number of people the census believes the entry represents.
- ~~**Education**~~
- **Education.Num**
- **Marital.Status**
- **Occupation**
- ~~**Relationship**~~
- **Race**
- **Sex**
- **Capital.Loss**
- **Capital.Gain:** Capital gains and losses are when you sell an asset for less or more than what you have invested it. It is mostly used for investments (i.e. property, securities, etc.). For instance, if you bought a stock for $10k and sold it for $20k, you'd have $10k in capital gains. Likewise if you sold that same stock for $5k, you'd have $5k in capital losses.

  Capital gains and losses are netted so that if you had $10k in capital gains and $20k in capital losses, those two would net to a $10k capital loss that would show up in the capital loss column. Therefore:

  data['capital.gain.loss'] = data['capital.gain'] - data['capital.loss']

- **Hours.Per.Week**
- **Native.Country**

Independent Variable

- **Label:** Whether the individual makes more than $50,000 annually (>$50k, <$50k).

Data Cleaning and Encoding

IMPORTING PACKAGES

Using pandas, numpy, sklearn, matplotlib to display plots, and xgboost to complete our model.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score, StratifiedKFold, trai
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
```

DATA CLEANING AND IMPUTATION

Take 'workclass' column for example, it reads 'NA' as '?'.

```
raw_data = pd.read_csv('adult.csv')
We already have "education_years", ''marital.status'',  columns that can c
```

Dropped multiple columns to eliminate redundancy or irrelevant information:

- **[education]:** The education number was redundant to another column of data.
- **[relationship]:** Relationship information was somewhat redundant and drilled down to a excessively granular level for what we are looking to accomplish.

```
```

```
pd.set_option('display.max_columns',None)
raw_data.drop(['education','relationship','Unnamed: 15','Unnamed: 16','Unn
workclass = raw_data['workclass'].replace("?",'Private')
raw_data['workclass'] = workclass
raw_data.replace('?','0',inplace = True)

raw_data.workclass.value_counts()
```

For the column that has obvious mode, we replace '?' with mode.

```
Out[30]: Private             22696
         Self-emp-not-inc     2541
         Local-gov            2093
         ?                    1836
         State-gov            1298
         Self-emp-inc         1116
         Federal-gov           960
         Without-pay            14
         Never-worked            7
         Name: workclass, dtype: int64
```

```
         Private             24532
         Self-emp-not-inc     2541
         Local-gov            2093
         State-gov            1298
         Self-emp-inc         1116
         Federal-gov           960
         Without-pay            14
         Never-worked            7
         Name: workclass, dtype: int64
```

Otherwise, distributed according to the proportion that taking out '?' (new category in the column)

```
row = raw_data['occupation'].value_counts()
row.drop(row.index[7],inplace = True)
summ = 0
for i  in row:
    summ += i
list_perc = []
for i, j in zip(row, row.index):
    list_perc.append(i/summ)
resultofo = np.random.choice(row.index,1843, p = list_perc )

```

```

raw_data.occupation.value_counts()
```

```
Prof-specialty       4140
Craft-repair         4099
Exec-managerial      4066
Adm-clerical         3770
Sales                3650
Other-service        3295
Machine-op-inspct    2002
?                    1843
Transport-moving     1597
Handlers-cleaners    1370
Farming-fishing       994
Tech-support          928
Protective-serv       649
Priv-house-serv       149
Armed-Forces            9
Name: occupation, dtype: int64
```

```
Prof-specialty       4369
Craft-repair         4347
Exec-managerial      4312
Adm-clerical         3998
Sales                3846
Other-service        3512
Machine-op-inspct    2140
Transport-moving     1691
Handlers-cleaners    1447
Farming-fishing      1048
Tech-support          987
Protective-serv       698
Priv-house-serv       156
Armed-Forces           10
Name: occupation, dtype: int64
```

ENCODING

Define x and y:

```
df_x = pd.get_dummies(raw_data,drop_first = True)
```

```
df_x.drop('income_>50K',axis = 1,inplace = True)
df_y = df_x.iloc[:,-1]
```

```
df_x
```

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week	workclass_Local-gov	workclass_Never-worked	workclass_Private	workclass_Self-emp-inc
0	90	77053	9	0	4356	40	0	0	1	0
1	82	132870	9	0	4356	18	0	0	1	0
2	66	186061	10	0	4356	40	0	0	1	0
3	54	140359	4	0	3900	40	0	0	1	0
4	41	264663	10	0	3900	40	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...
32556	22	310152	10	0	0	40	0	0	1	0
32557	27	257302	12	0	0	38	0	0	1	0
32558	40	154374	9	0	0	40	0	0	1	0
32559	58	151910	9	0	0	40	0	0	1	0
32560	22	201490	9	0	0	20	0	0	1	0

32561 rows × 37 columns

```
df_y
```

```
0         0
1         0
2         0
3         0
4         0
         ..
32556     0
32557     0
32558     1
32559     0
32560     0
Name: income_>50K, Length: 32561, dtype: uint8
```

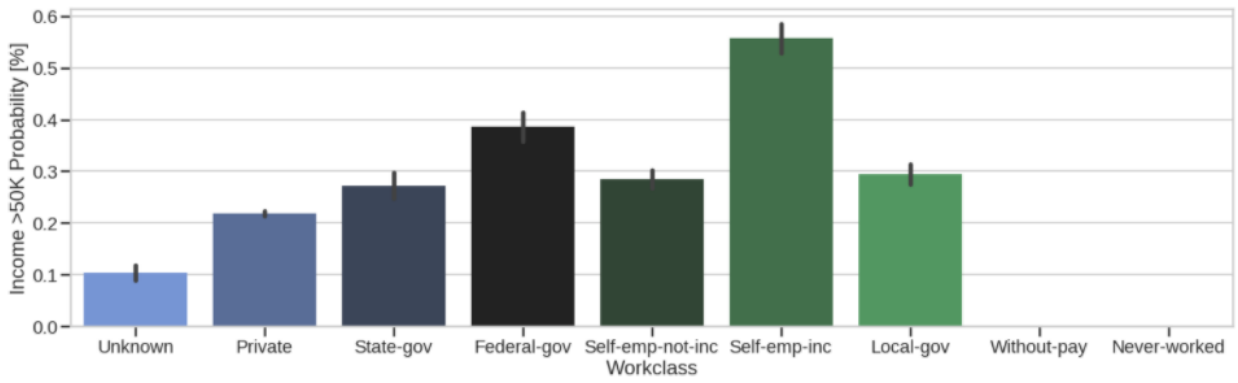
Data visualization

Barplot of workclass vs. income

```
dataset['income']=dataset['income'].map({'<=50K': 0, '>50K': 1})
dataset["workclass"] = dataset["workclass"].replace(["?"],'Unknown')
fig, ax = plt.subplots(figsize=(25,7))
sns.set_context("poster")
current_palette = sns.diverging_palette(255, 133, l=60, n=7, center="dark"

fig = sns.barplot(x='workclass',y='income',data=dataset,palette=current_pa

fig.set_ylabel("Income >50K Probability [%]")
fig.set_xlabel("Workclass")
```

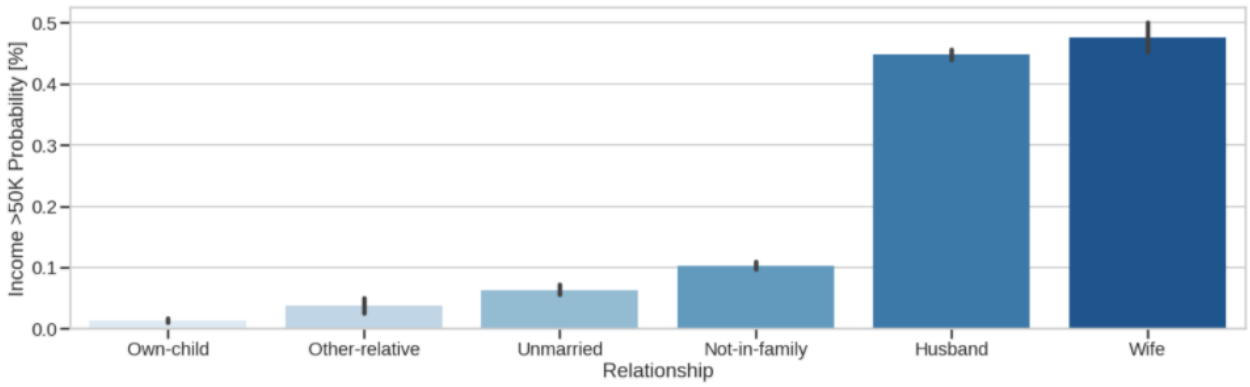


Barplot of Relationship vs Income

```
fig, ax = plt.subplots(figsize=(25,7))
sns.set_context("poster")
current_palette = sns.color_palette("Blues")

fig = sns.barplot(x='relationship',y='income',data=dataset, order=['Own-ch

fig.set_ylabel("Income >50K Probability [%]")
fig.set_xlabel("Relationship")
```

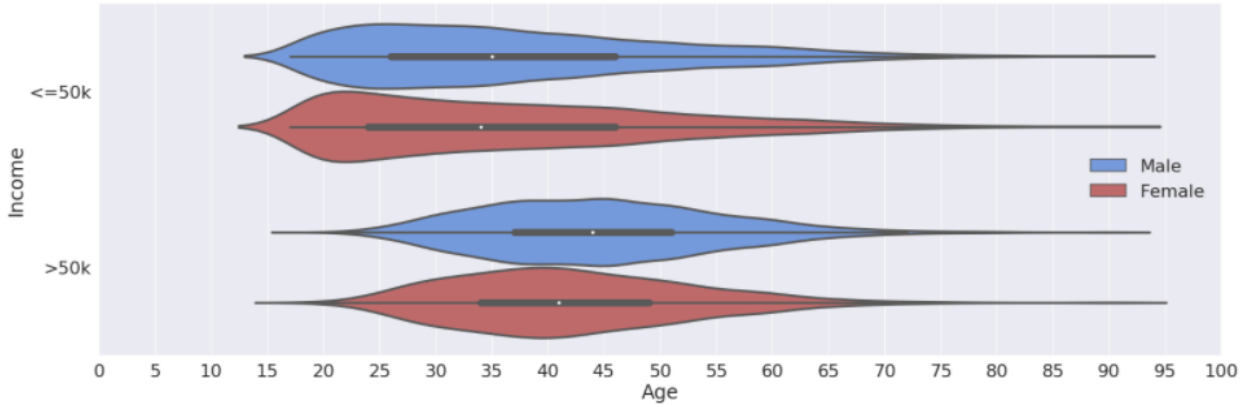


Violinplot of age vs sex vs income

```
fig, axe = plt.subplots(figsize=(25,8))
sns.set_context("poster")

g=sns.violinplot(x='age',y='income',hue='sex',hue_order=["Male","Female"],data=dataset,ori
ent="h",palette=["cornflowerblue","indianred",])

g.set_ylabel("Income")
g.set_xlabel("Age")
g.set(yticklabels =['<=50k','>50k'])
setThis = g.legend(loc='center right')
plt.xlim(0,100)
axe.xaxis.set_major_locator(ticker.MultipleLocator(5))
```



Correlation between X and Y

```
corr_df = pd.DataFrame(np.zeros([2,df_x.shape[1]]))
```

```
for j in range(df_x.shape[1]):
    A = pd.concat([df_y,df_x.iloc[:,[j]]], axis = 1)
    vol =  A.corr().iloc[0,1]
    corr_df.iloc[:,j] = A.columns[1],vol
corr_df.columns = corr_df.iloc[0,:]
corr_df.index  = [0,"income_50K"]
corr_df.drop(0,inplace = True)
```

```
corr_df
```

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week	workclass_Local-gov	workclass_Never-worked	workclass_Private	workclass en
income_50K	0.234037	-0.00946256	0.335154	0.223329	0.150526	0.229689	0.0330906	-0.00825864	-0.125573	0.1

Stats number for the data

stats for x

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week	workclass_Local-gov	workclass_Never-worked	workclass_Private
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456	0.064279	0.000215	0.753417
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429	0.245254	0.014661	0.431029
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000	0.000000	0.000000	1.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000	0.000000	0.000000	1.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000	0.000000	0.000000	1.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000	1.000000	1.000000	1.000000

stats for y

```
count    32561.000000
mean         0.240810
std          0.427581
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: income_>50K, dtype: float64
```

Modeling

1. Random Forest

```
#random forest
for i in n_estimators:
    RFC = RandomForestClassifier(n_estimators = i,max_features = 'auto')
    kfoldCV = KFold(n_splits = 5)
    result2 = cross_val_score(RFC,X_train, y_train, cv=kfoldCV)
    print(f'Accuracy: %f ---- ' %result2.mean(), 'n-estimators = %d' %i)

```

```
Accuracy: 0.854922 ----  n-estimators = 50
Accuracy: 0.856112 ----  n-estimators = 150
Accuracy: 0.856150 ----  n-estimators = 300
Accuracy: 0.856688 ----  n-estimators = 450
Accuracy: 0.856112 ----  n-estimators = 600
```

---

2. XGBoost classifier (Gradient decent)

```
#random state use for initialize status for internal random number
X_train, X_test, y_train, y_test = train_test_split(df_x, df_y, test_size=
kfoldCV = StratifiedKFold(n_splits=5)
n_estimators  = [50,150,300,450,600]
for i in n_estimators:
    xgb_model = XGBClassifier(n_estimators=i)
    results = cross_val_score(xgb_model, X_train, y_train, cv=kfoldCV)
    print(f'Accuracy: %f ---- ' %results.mean())
```

a) classification report

```
#predict on test data
xgb_model.fit(X_train, y_train)
prediction = xgb_model.predict(X_test)
```

```
summary = classification_report(y_test,prediction)
print(summary)
```

	precision	recall	f1-score	support
0	0.89	0.93	0.91	4965
1	0.73	0.65	0.69	1548
accuracy			0.86	6513
macro avg	0.81	0.79	0.80	6513
weighted avg	0.86	0.86	0.86	6513

precision is that in the data we test positive, how many positive data are actually predicted by us. The result still need to be further improved.

---

3. LOGISTIC REGRESSION

```
#logistic regression
logit = LogisticRegression(solver='lbfgs')
kfoldCV = KFold(n_splits = 5)
result3 = cross_val_score(logit,X_train, y_train, cv=kfoldCV)
print(f'Accuracy: %f ---- ' %result3.mean())
```

Accuracy: 0.797182 ----

---

4. Gradient Boosting

1.Train the model and then test the model

```
X_train, X_test, y_train, y_test = train_test_split(df_x, df_y, test_size=

for trees in [20, 100, 200, 500]:
    for learnin_rate in [0.1, 0.2, 0.5]:
```

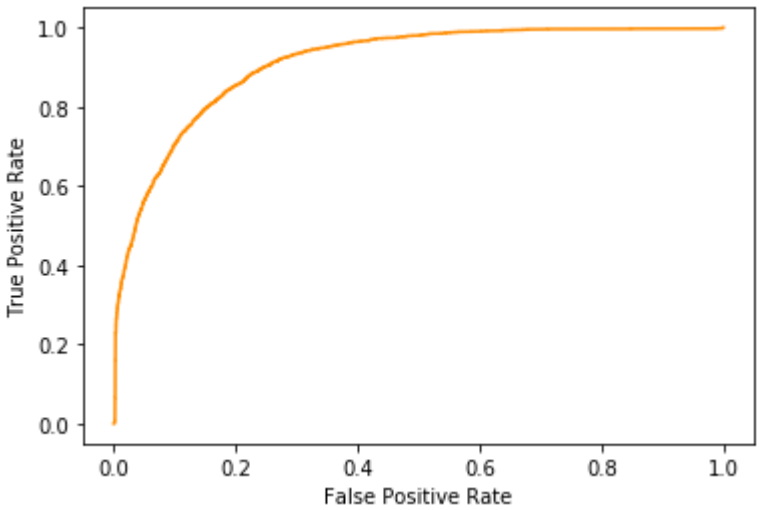
```
        gb_clf = GradientBoostingClassifier(n_estimators=trees, learning_r
                                            max_depth=4, random_state=0)
gb_clf.fit(X_train, y_train)

predicted_values = gb_clf.predict_proba(X_test)[:,1]
```

2.Plot ROC curve and calculate AUC

```
fpr, tpr, thresholds  =  roc_curve(y_test, predicted_values)

plt.plot(fpr, tpr, color='darkorange')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```



ROC curve

```
roc_auc_score(y_test, predicted_values)
```

0.9105270732698724

---

5. Decision Tree

```
#decision tree
clf = DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
predicted_v = clf.predict_proba(X_test)[:,1]
```

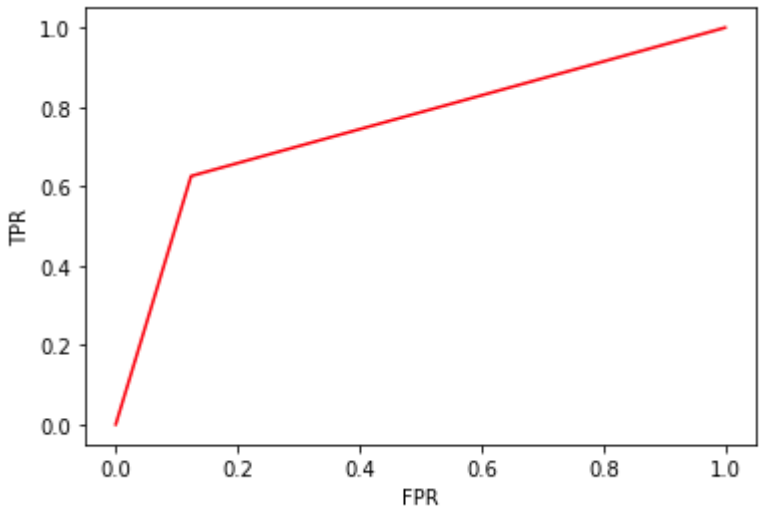
```
roc_auc_score(y_test, predicted_v)
```

0.7511516663153703

R.O.C Curve

```
FPR, TPR ,thresh = roc_curve(y_test, predicted_v)

plt.plot(FPR,TPR, color = 'red')
plt.xlabel('FPR')
plt.xlabel("TPR")
```



Reference

The discuss shows the method used in logit regression and optimization issues.

https://stackoverflow.com/questions/45850841/does-sklearn-linear-model-logisticregression-always-converge-to-best-solution

---

The web side shows how XGBoost work

https://towardsdatascience.com/how-does-xgboost-work-748bc75c58aa

FAQ

Is it possible to call columns by index?

Have you had a chance to answer the previous question?

Yes, after a few months we finally found the answer. Sadly, Mike is on vacations right now so I'm afraid we are not able to provide the answer at this point.

YC:

recommend drop native.country column.

I found if we use zero to replace ?, it will create a new category. Maybe use mode instead would be good.

https://www.kaggle.com/mattmet/income-prediction-xgboost-accuracy-86-02

Regarding final projects; first, as mentioned, do something that you can proudly present to a potential employer. So, write a neat, professional report on what you have done. At a minimum, include following sections in your report:

1. Business context. How this model will solve a problem.

2. An explanation of data. Add some summary stats and graphs here.

3. Data preparation (missing imputation, encoding, normalization, ...)

4. Sampling (test and train)

5. Modeling process

6. Performance evaluation and final model

Best,

Amir

NN----------

info from 11/16 20 min presentation

How does the NN work?

what is 15 in unit?

crossentrophy?

what is grid-research? parameter testing and finding solution.

For correlation, classification work the same way? professor.

what is the difference between XGBclassifier, GBM.

11/19

activation= 'sigmoid'?????? how about other methods?

adding layers? unit? influence

numeric and classification?

What is batch size in neural network?

https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network

GridsearchCV-

roc_auc_score(y_test, y_pred) is too low, should improve

N N output2

one layer, unit = 16

```
Epoch 1/20
869/869 [==============================] - 0s 567us/step - loss: 3388.1294 - accuracy: 0.6651
Epoch 2/20
869/869 [==============================] - 1s 597us/step - loss: 0.5940 - accuracy: 0.7654
Epoch 3/20
869/869 [==============================] - 0s 572us/step - loss: 0.5590 - accuracy: 0.7611
Epoch 4/20
869/869 [==============================] - 1s 603us/step - loss: 0.5579 - accuracy: 0.7619
Epoch 5/20
869/869 [==============================] - 1s 614us/step - loss: 0.5534 - accuracy: 0.7619
Epoch 6/20
869/869 [==============================] - 1s 606us/step - loss: 0.5577 - accuracy: 0.7611
Epoch 7/20
869/869 [==============================] - 1s 620us/step - loss: 0.5533 - accuracy: 0.7595
Epoch 8/20
869/869 [==============================] - 1s 604us/step - loss: 0.5526 - accuracy: 0.7587
Epoch 9/20
869/869 [==============================] - 1s 602us/step - loss: 0.5526 - accuracy: 0.7587
Epoch 10/20
869/869 [==============================] - 1s 636us/step - loss: 0.5526 - accuracy: 0.7587
Epoch 11/20
869/869 [==============================] - 1s 619us/step - loss: 0.5526 - accuracy: 0.7587
Epoch 12/20
869/869 [==============================] - 1s 615us/step - loss: 0.5526 - accuracy: 0.7587
Epoch 13/20
869/869 [==============================] - 1s 603us/step - loss: 0.5525 - accuracy: 0.7587
Epoch 14/20
869/869 [==============================] - 1s 610us/step - loss: 0.5526 - accuracy: 0.7587
Epoch 15/20
869/869 [==============================] - 1s 610us/step - loss: 0.5526 - accuracy: 0.7587
Epoch 16/20
869/869 [==============================] - 1s 607us/step - loss: 0.5526 - accuracy: 0.7587
Epoch 17/20
869/869 [==============================] - 1s 617us/step - loss: 0.5526 - accuracy: 0.7587
Epoch 18/20
869/869 [==============================] - 1s 616us/step - loss: 0.5526 - accuracy: 0.7587
Epoch 19/20
869/869 [==============================] - 1s 607us/step - loss: 0.5526 - accuracy: 0.7587
Epoch 20/20
869/869 [==============================] - 1s 621us/step - loss: 0.5526 - accuracy: 0.7587
```

One layer, unit = 5

```
Epoch 1/20
869/869 [==============================] - 1s 596us/step - loss: 97.7203 - accuracy: 0.6537
Epoch 2/20
869/869 [==============================] - 1s 637us/step - loss: 18.8603 - accuracy: 0.6790
Epoch 3/20
869/869 [==============================] - 1s 629us/step - loss: 15.4449 - accuracy: 0.6777
Epoch 4/20
869/869 [==============================] - 1s 608us/step - loss: 21.2471 - accuracy: 0.6785
Epoch 5/20
869/869 [==============================] - 1s 611us/step - loss: 14.8098 - accuracy: 0.6832
Epoch 6/20
869/869 [==============================] - 1s 613us/step - loss: 13.6507 - accuracy: 0.6868
Epoch 7/20
869/869 [==============================] - 1s 601us/step - loss: 15.5515 - accuracy: 0.6887
Epoch 8/20
869/869 [==============================] - 1s 605us/step - loss: 14.3791 - accuracy: 0.6887
Epoch 9/20
869/869 [==============================] - 1s 599us/step - loss: 11.7765 - accuracy: 0.6952 0s - loss: 11.5446 - accu
racy: 0
Epoch 10/20
869/869 [==============================] - 1s 602us/step - loss: 11.0546 - accuracy: 0.6985
Epoch 11/20
869/869 [==============================] - 1s 608us/step - loss: 10.6102 - accuracy: 0.7020
Epoch 12/20
869/869 [==============================] - 1s 606us/step - loss: 10.3619 - accuracy: 0.7004
Epoch 13/20
869/869 [==============================] - 1s 596us/step - loss: 12.1036 - accuracy: 0.6987
Epoch 14/20
869/869 [==============================] - 1s 591us/step - loss: 9.0637 - accuracy: 0.7046
Epoch 15/20
869/869 [==============================] - 1s 595us/step - loss: 9.9866 - accuracy: 0.7059
Epoch 16/20
869/869 [==============================] - 1s 589us/step - loss: 10.0994 - accuracy: 0.7076
Epoch 17/20
869/869 [==============================] - 1s 607us/step - loss: 8.9119 - accuracy: 0.7100
Epoch 18/20
869/869 [==============================] - 1s 595us/step - loss: 8.4940 - accuracy: 0.7168
Epoch 19/20
869/869 [==============================] - 1s 595us/step - loss: 8.2081 - accuracy: 0.7131
Epoch 20/20
869/869 [==============================] - 1s 600us/step - loss: 8.1296 - accuracy: 0.7164
```