hw5a

October 30, 2023

1 CSC380 Homework 5

INDIVIDUAL HOMEWORK The homework is not collaborative anymore. Please respect the academic integrity. **Remember: if you get caught on cheating, you get F.**

Each subproblem is worth 10 pts. All cells are marked with instructions to insert your code. Please complete all cells as directed.

What to turn in: - Please print the notebook containing the answers and results into a pdf file (you can use `File - Print`). Submit this pdf file to the main homework entry in gradescope. Be sure to locate your answers for each problem when you submit, as ususal. In the worst case where you cannot print it into a pdf file somehow, you can create a Microsoft word document and then copy-paste screenshots showing your code and output parts by parts. - You also need to submit this jupyter notebook file filled with your answers in the code entry in gradescope.

2 Problem 1 : K-Nearest Neighbor classification

3 To Show or Not To Show the ad? - Targetted Advertising

Free Social Media platforms like Facebook, make's a lion's share of total revenue, from ads, specifically targetted ads. Last year alone, it rose from 46% from a year earlier to $25.44 billion.

Targeted advertising is advertising content to customers based on their interests, traits, and behaviors. Instead of pushing an ad to general public, hoping some of them buy your product, you show your ads only to those people who are likely to buy it the first place. Less money into advertising, more money made in Sales.

In this problem, you are a data scientist at this new up and coming social media platform. We want advertisers on our platform, to perform well in advertising. So we are going to run targeted ads. We have details of our users who already bought product X after seeing the ads. The dataset used is a modified version of Social Network Ads DataSet

In this problem, we are going to use **K-Nearest Neighbor classification** to find potential customers to show ads of a product X. Given the details about a person, how likely are they to buy the product? We will use these predictions to guide our target audience for ads.

[189]:
```
import warnings
# Suppress warnings
warnings.filterwarnings("ignore")
```

3.1 Data

The first step is to look into the data : Here, since the data do not have any missing values, we deal with two other issues.

1. Category Encoding
2. Feature Scaling

[190]:
```
import pandas as pd
```

[191]:
```
purchases_df = pd.read_csv('data/train_ads.csv')
purchases_df.head()
```

[191]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

We have three features that would be super useful to make the decision. Gender, Age, and Estimated Salary. Our Target is the column Purchased.

[192]:
```
X_train = purchases_df[['Gender','Age','EstimatedSalary']]
Y_train = purchases_df['Purchased']
```

3.1.1 a. Encode categorical values

Our data contains a catgorical feature (Gender) which is a string. In order to train the model we must first convert the categorical data into numerical values. The Scikit-Learn LabelEncoder class handles this for you. Please use the Preprocessing.LabelEncoder.fit_transform() function to fit and transform categorical values to numerical values for Gender. The procedure is similar to what you performed in Problem 1.

Documentation - Scikit-Learn - LabelEncoder

[193]:
```
# Insert your code here
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
purchases_df['Gender'] = le.fit_transform(purchases_df['Gender'])
X_train = purchases_df[['Gender','Age','EstimatedSalary']]
X_train
```

[193]:

	Gender	Age	EstimatedSalary
0	1	19	19000
1	1	35	20000
2	0	26	43000
3	0	27	57000
4	1	19	76000
..	…	…	…

```
375        0   46             32000
376        0   46             74000
377        0   42             53000
378        1   41             87000
379        0   58             23000

[380 rows x 3 columns]
```

3.1.2 b. Feature Scaling

Feature Scaling is a technique to standardize the independent features present in the data. The Scikit-Learn Preprocessing StandardScaler performs Z-scoring, as discussed in lecutre. This will help all numerical data satisfy mean 0 and variance 1. Without this, features with higher variance may dominate the predictions. Please use StandardScalar to fit and transform all of your data.

sklearn.preprocessing.StandardScaler

[194]:
```
# Insert your code here
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
# scaler.fit_transform(X_train, Y_train)
# print(scaler.fit(purchases_df[['Gender']]))
# purchases_df['Gender'] = scaler.transform(purchases_df[['Gender']])
# purchases_df['Gender']
# print(scaler.transform(purchases_df[['Gender','Age','EstimatedSalary',␣
↪'Purchased']]))
```

3.2 Model Training, Selection, and Evaluation

We will now fit our K-Nearest Neighbor model to training data, compare variations on the model, and evaluate our best model on Test data.

3.2.1 c. Initial Model Fit

In the cell below create a KNeighborsClassifier object with K=2 neighbors, and evaluate cross-validation (CV) score (using cross_val_score) with 10 folds. Report the mean and standard deviation of the accuracy CV score.

sklearn.neighbors.KNeighborsClassifier

[195]:
```
# Insert your code here
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
import numpy as np
neigh = KNeighborsClassifier(n_neighbors=2)
neigh.fit(X_train, Y_train)
neigh.score(X_train, Y_train)
cv_score = cross_val_score(neigh, X_train, Y_train, cv = 10)
```

```
print("CV score with 10 folds:")
print(cv_score)
print("CV score mean:")
print(np.mean(cv_score))
print("CV score std:")
print(np.std(cv_score))
```

```
CV score with 10 folds:
[0.76315789 0.76315789 0.89473684 0.92105263 0.86842105 0.81578947
 0.86842105 0.73684211 0.76315789 0.89473684]
CV score mean:
0.8289473684210528
CV score std:
0.06472828355920444
```

3.2.2 d. Model Selection

Recall that a K-Nearest Neighbor model has a single *hyperparameter*: the number of neighbors K. We will perform model selection in order to choose the optimal value of K. To do this, evaluate the cross validation score (10-fold) for 15 models, each model having a different value K in the range 1 to 15. Produce the following output: * A plot of K versus accuracy (label axes and title your plot) * Print the value of the highest accuracy * Print the value of K that achieves the highest accuracy

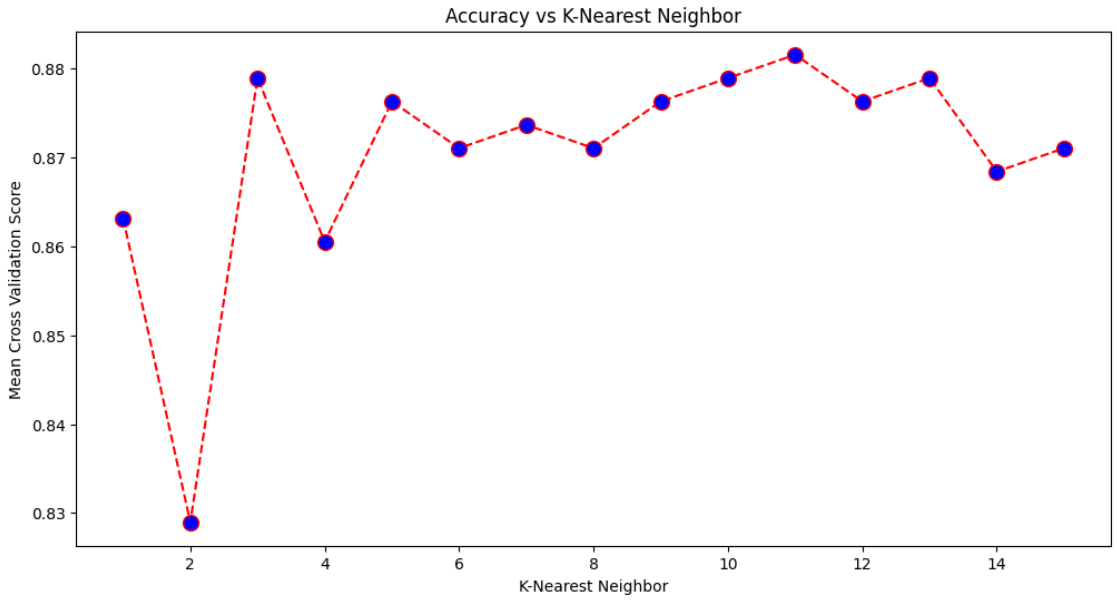
As a general rule, when there is a tie, choose the one that induces smoother decision boundary (larger K in our case); otherwise, points will be taken off.

This is a great resource Finding the optimal value of k

[196]:
```
# Insert your code here
accuracy = []
for i in range (1, 16):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, Y_train)
    cv_score = cross_val_score(knn, X_train, Y_train, cv = 10)
    mean = np.mean(cv_score)
    accuracy.append(mean)

import matplotlib.pyplot as plt
plt.figure(figsize=(12,6))
plt.plot(range(1, 16), accuracy, color = 'red', linestyle='dashed', marker='o',␣
 ↪markerfacecolor='blue', markersize=10)
plt.title("Accuracy vs K-Nearest Neighbor")
plt.xlabel('K-Nearest Neighbor')
plt.ylabel('Mean Cross Validation Score')
print('Highest accuracy:')
print(max(accuracy))
print('Value of K for highest accuracy:')
print(accuracy.index(max(accuracy)) + 1)
```

```
Highest accuracy:
0.881578947368421
Value of K for highest accuracy:
11
```



3.2.3 e. Evaluate the Model

Now, you will a more careful evaluation of your selected model. Unlike cross_val_score, which only allows a single scoring function, cross_validate accepts a tuple of scoring functions. In cell below please create a KNeighborsClassifier with the optimal K value as chosen above. Use the model_selection.cross_validate function to perform 10-fold cross validation and report the all of the following scores:

- Average Prediction accuracy
- Average Precision
- Average Recall
- Average F1

Please make the output as readible as possible for your graders.

Documentation - SciKit-Learn - model_selection.cross_validate

[201]:
```
# Insert your code here
knn = KNeighborsClassifier(n_neighbors=11)
knn.fit(X_train, Y_train)
from sklearn.model_selection import cross_validate
from sklearn import metrics
cv_results = cross_validate(knn, X_train, Y_train, cv = 10,␣
 ↪scoring=['accuracy', 'precision', 'recall', 'f1'], return_train_score=True)
```

```
cv_results
print("Average prediction accuracy:")
print(np.mean(cv_results['train_accuracy']))
print("Average precision accuracy:")
print(np.mean(cv_results['train_precision']))
print("Average recall accuracy:")
print(np.mean(cv_results['train_recall']))
print("Average f1 accuracy:")
print(np.mean(cv_results['train_f1']))
```

```
Average prediction accuracy:
0.9090643274853802
Average precision accuracy:
0.8510795595835929
Average recall accuracy:
0.8827765064836003
Average f1 accuracy:
0.8665318190061153
```

3.2.4 f. Train the Final Model

Cross validation doesn't produce a fitted model–that isn't its purpose. Instead, cross-validation is used to estimate the generalization scores by averaging scores across multiple train/validation splits. In the cell below we will finally train our selected model. Do the following: * Create a K-Nearest Neighbors model with the optimum K value (as we chose previously) * Train the model on all training data (do not use cross validation) * Report prediction accuracy on the training set * Compute and report the confusion matrix on the training data (there is a sklearn function for it)

Explanation on confusion matrix: One useful statistic for evaluating classifiers is the *confusion matrix*, which enumerates categories of correct and incorrect classifications. For more information see the Wikipedia article on the confusion matrix. Compute the confusion matrix and report whether one class is confused for another class more often, or whether they are about the same (within a couple of points).

Documentation - Scikit-Learn - metrics.confusion_matrix

[222]:
```
# Insert your code here
knn = KNeighborsClassifier(n_neighbors=11)
knn.fit(X_train, Y_train)
print("Prediction accuracy of the training set:")
print(knn.score(X_train, Y_train))
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_train, knn.predict(X_train))
```

```
Prediction accuracy of the training set:
0.9078947368421053
```

[222]:
```
array([[233,  20],
       [ 15, 112]])
```

3.3 Testing the Model

You may have noticed that we did not ask you to create a Train / Test split. This dataset has a standard Training / Test split. However, in this dataset, the Test data represet actual user features who have not been served ads. As a result, we do not have labels in the true Test set and so cannot compute test accuracy. The following cell loads the test data, which you will then evaluate using a variety of metrics.

[210]:
```
# Insert your code here
test_df = pd.read_csv("data/test_ads.csv")
```

3.4 g. Applying the Model

Now that we have build our model, let's look for potensial customers of product X to send the ads too. In the data, you will find a csv test_ads.csv.

Read the csv, perfrom the Categorical Encoding and Feature Scaling. **Remember to use the same Encoder and Standard Scalar as the ones you used above** and print the userIds of users who are likely to buy the product. part of the work has been done for you.

[228]:
```
# Insert your code here
test_df['Gender'] = le.fit_transform(test_df['Gender'])
X_test = test_df[['Gender','Age','EstimatedSalary']]
X_test = scaler.fit_transform(X_test)
prediction = knn.predict(X_test)
print("User IDs of users likely to by the product:")
for i in range(len(prediction)):
    if (prediction[i] == 1):
        print(test_df['User ID'][i])
```

```
User IDs of users likely to by the product:
15715622
15806901
15775335
15635893
```

4 Conclusion

Congragulations, Thanks to your Data Science Skills, our new social media platform is doing well in advertising, and the Clients sales skyrocketed! Your customers are happier and buissness is now booming !

HW5b

October 30, 2023

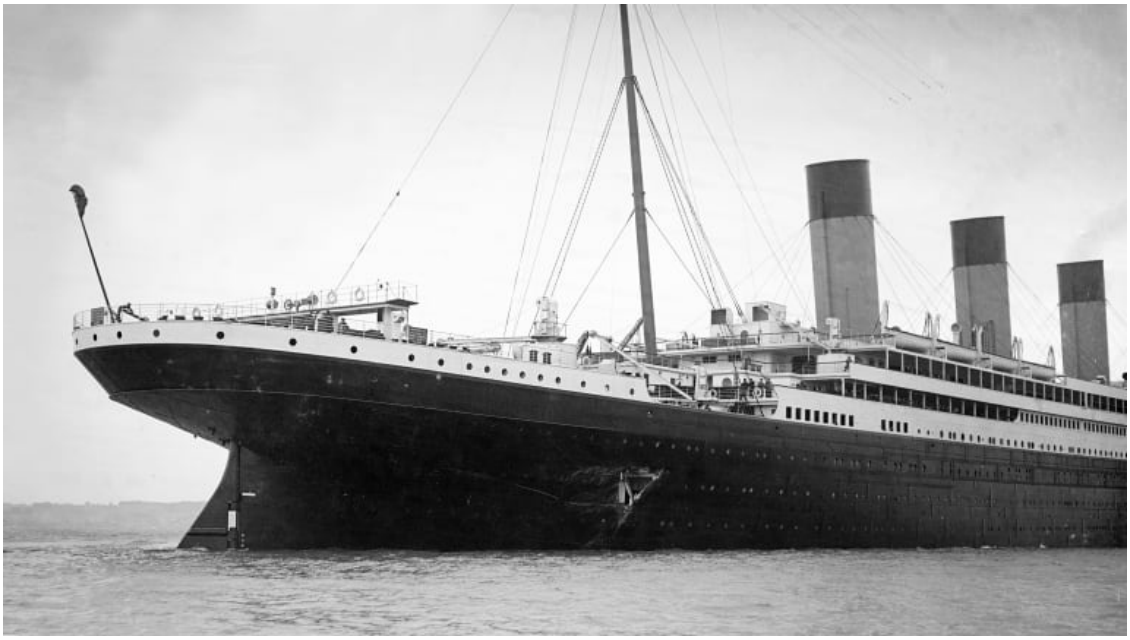
1 CSC380 Homework 5 : Problem 2 : Naive Bayes Classifier

1.1 Did They Survive the Sinking of the Titanic?

The RMS Titanic was a passenger ship that infamously sunk during its maiden voyage in 1912. Most of the people on the ship unfortunately perished. In this problem we are going to train a **Naive Bayes Classifier** on features the passengers to see if we can accurately predict whether passengers survived the disaster. The features of each passenger include the fare they paid, their age, gender, etc. We emphasize that the model does not learn *causal* relationships, it only learns which features *correlate* with survival in the data. **This is just for fun**: do not draw any conclusions from this analysis. This dataset was part of a popular Kaggle Competition. If you're curious you may enjoy a quick video about the competition.

1.1.1 What to Submit

Follow the instruction from problem 1.



[73]:
```
import warnings
# Suppress warnings
warnings.filterwarnings("ignore")
```

1.2 Data

The first steps in any data science project involve loading and cleaning up data. In our example we will need to deal with two issues: handling missing values and converting categorical data into numerical quantities that can be handled by the Naive Bayes model. We will start by loading the data.

[74]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = pd.read_csv('data/titanic.csv')

# The features we will use
features = ['Age', 'Embarked', 'Fare', 'Parch', 'Pclass', 'Sex', 'SibSp']
X = data[features]

# The target (predictor)
target = 'Survived'
Y = data[target]

# replace missing values
X["Age"].fillna(np.random.choice(X['Age'][~X['Age'].isna()]),inplace = True)
X["Embarked"].fillna(np.random.choice(X['Embarked'][~X['Embarked'].
 ↪isna()]),inplace = True)
X.head(30)
```

[74]:

	Age	Embarked	Fare	Parch	Pclass	Sex	SibSp
0	22.0	S	7.2500	0	3	male	1
1	38.0	C	71.2833	0	1	female	1
2	26.0	S	7.9250	0	3	female	0
3	35.0	S	53.1000	0	1	female	1
4	35.0	S	8.0500	0	3	male	0
5	29.0	Q	8.4583	0	3	male	0
6	54.0	S	51.8625	0	1	male	0
7	2.0	S	21.0750	1	3	male	3
8	27.0	S	11.1333	2	3	female	0
9	14.0	C	30.0708	0	2	female	1
10	4.0	S	16.7000	1	3	female	1
11	58.0	S	26.5500	0	1	female	0
12	20.0	S	8.0500	0	3	male	0
13	39.0	S	31.2750	5	3	male	1
14	14.0	S	7.8542	0	3	female	0
15	55.0	S	16.0000	0	2	female	0
16	2.0	Q	29.1250	1	3	male	4
17	29.0	S	13.0000	0	2	male	0
18	31.0	S	18.0000	0	3	female	1
19	29.0	C	7.2250	0	3	female	0

20	35.0	S	26.0000	0	2	male	0
21	34.0	S	13.0000	0	2	male	0
22	15.0	Q	8.0292	0	3	female	0
23	28.0	S	35.5000	0	1	male	0
24	8.0	S	21.0750	1	3	female	3
25	38.0	S	31.3875	5	3	female	1
26	29.0	C	7.2250	0	3	male	0
27	19.0	S	263.0000	2	1	male	3
28	29.0	Q	7.8792	0	3	female	0
29	29.0	S	7.8958	0	3	male	0

1.2.1 a. Encode categorical values

Our data contains two categorical features, "Sex" and "Embarked", both containing string values. In order to train a Naive Bayes model we must first convert these categorical data into numerical values. The Scikit-Learn LabelEncoder class handles this for you. Please use the Preprocessing.LabelEncoder.fit_transform() function to fit and transform categorical values to numerical values for, both, the "Sex" and "Embarked" fields.

Documentation - Scikit-Learn - LabelEncoder

[75]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()#instaiate the the Label Encoder here.
X['Sex'] = le.fit_transform(X['Sex'])#insert your code here
X['Embarked'] = le.fit_transform(X['Embarked'])#insert your code here
X.head(5)
```

[75]:

	Age	Embarked	Fare	Parch	Pclass	Sex	SibSp
0	22.0	2	7.2500	0	3	1	1
1	38.0	0	71.2833	0	1	0	1
2	26.0	2	7.9250	0	3	0	0
3	35.0	2	53.1000	0	1	0	1
4	35.0	2	8.0500	0	3	1	0

1.2.2 b. Split the dataset into test and train

Use the scikit-learn "train_test_split" function to create a Training / Test split with **75%** of the data designated to training, and **25%** to testing. Make sure to use the random state provided below, to ensure that everyone has the same training/test split.

Documentation - Scikit-Learn - Train Test Split

[76]:
```
from sklearn.model_selection import train_test_split
random_state=42
test_size=0.25

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =␣
↪test_size, random_state = random_state)#insert your code here
```

```
# look at some training data
X_train.head(5)
```

[76]:

	Age	Embarked	Fare	Parch	Pclass	Sex	SibSp
298	29.00	2	30.5000	0	1	1	0
884	25.00	2	7.0500	0	3	1	0
247	24.00	2	14.5000	2	2	0	0
478	22.00	2	7.5208	0	3	1	0
305	0.92	2	151.5500	2	1	1	1

1.2.3 c. Summary statistics

It's good to make sure that the statistics of your data in training and test are similar. The Pandas "DataFrame.describe()" function is a very useful function for computing summary statistics of a DataFrame. Use the "describe" function on the training and test features (X_train and X_test) to look at their summary statistics.

Documentation - Pandas - DataFrame.describe

[77]:
```
X_train.describe()#insert your code here
```

[77]:

	Age	Embarked	Fare	Parch	Pclass	Sex	SibSp
count	668.000000	668.000000	668.000000	668.000000	668.000000	668.000000	668.000000
mean	29.338084	1.562874	32.179397	0.372754	2.333832	0.657186	0.553892
std	13.010575	0.772813	51.604012	0.795588	0.823707	0.475006	1.185279
min	0.420000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
25%	22.000000	1.000000	7.925000	0.000000	2.000000	0.000000	0.000000
50%	29.000000	2.000000	14.400000	0.000000	3.000000	1.000000	0.000000
75%	35.000000	2.000000	30.500000	0.000000	3.000000	1.000000	1.000000
max	80.000000	2.000000	512.329200	6.000000	3.000000	1.000000	8.000000

[78]:
```
X_test.describe()#insert your code here
```

[78]:

	Age	Embarked	Fare	Parch	Pclass	Sex	\
count	223.000000	223.000000	223.000000	223.000000	223.000000	223.000000	
mean	30.225695	1.457399	32.278530	0.408072	2.233184	0.618834	
std	12.994763	0.841880	43.578316	0.837912	0.869593	0.486766	
min	0.830000	0.000000	0.000000	0.000000	1.000000	0.000000	

```
25%     23.000000    1.000000    7.895800    0.000000    1.000000    0.000000
50%     29.000000    2.000000   15.245800    0.000000    3.000000    1.000000
75%     36.000000    2.000000   31.331250    1.000000    3.000000    1.000000
max     71.000000    2.000000  262.375000    5.000000    3.000000    1.000000
```

```
            SibSp
count  223.000000
mean     0.430493
std      0.801667
min      0.000000
25%      0.000000
50%      0.000000
75%      1.000000
max      4.000000
```

1.3 Checking Model Assumptions

Recall from lecture that Niave Bayes models features as conditionally independent, given the class label. In a real data this assumption doesn't hold. It's always good to test your assumptions to see how badly they are violated in the data. One way to test for (linear) dependence is to measure correlation. You will compute the Pearson correlation coefficient on each pair of features to give us a hint about how independent they are from the others.

[79]:
```
import numpy as np
#Some intialisations

columns = features +[target]
nColumns = len(columns)

result = pd.DataFrame(np.zeros((nColumns, nColumns)), columns=columns)

train = X_train.copy()
train[target]= Y_train
```

[80]:
```
from scipy.stats.stats import pearsonr
# Apply Pearson correlation on each pair of features.
for col_a in range(nColumns):
    for col_b in range(nColumns):
        result.iloc[[col_a], [col_b]] = round(pearsonr(train.loc[:,␣
 ↪columns[col_a]], train.loc[:,  columns[col_b]])[0],2)
```

[81]:
```
fig, ax = plt.subplots(figsize=(10,10))
im = ax.imshow(result)

#,yticklabels=columns, vmin=-1, vmax=1, annot=True, fmt='.2f', linewidths=.2)

# We want to show all ticks...
```

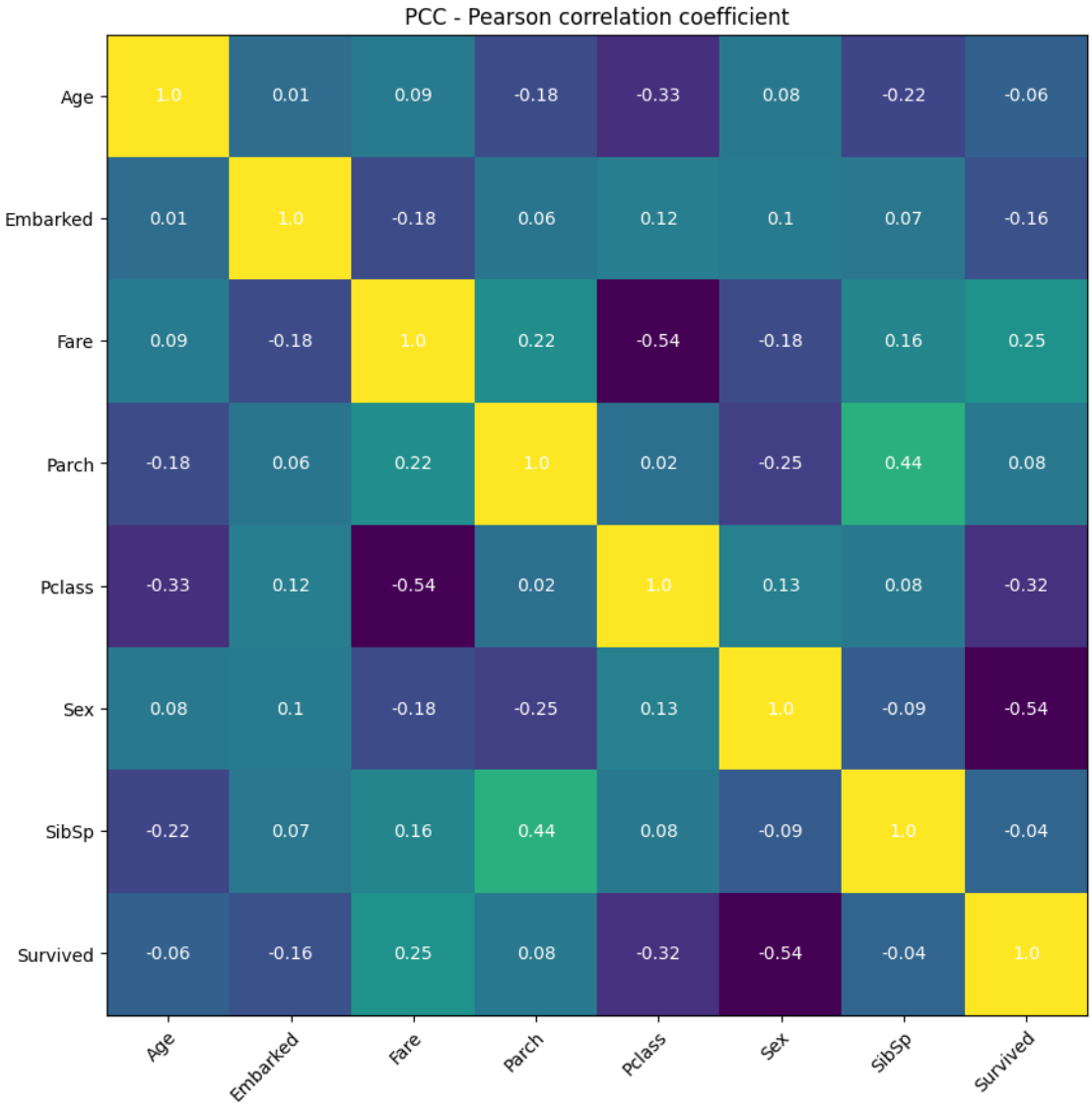
```
ax.set_xticks(np.arange(nColumns))
ax.set_yticks(np.arange(nColumns))

# ... and label them with the respective list entries
ax.set_xticklabels(columns)
ax.set_yticklabels(columns)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")

# Loop over data dimensions and create text annotations.
for i in range(nColumns):
    for j in range(nColumns):
        text = ax.text(j, i, result.iloc[i, j],ha="center", va="center",␣
 ↪color="w")

ax.set_title('PCC - Pearson correlation coefficient')
plt.show()
```



1.3.1 d. Normal feature assumption

We will model the class-conditional distributions of our continuous numerical features as Normal distributions. Let's check that assumption as well. In the cell below, use the DataFrame.plot(kind='density') function to plot the densities of each of your numerical features.

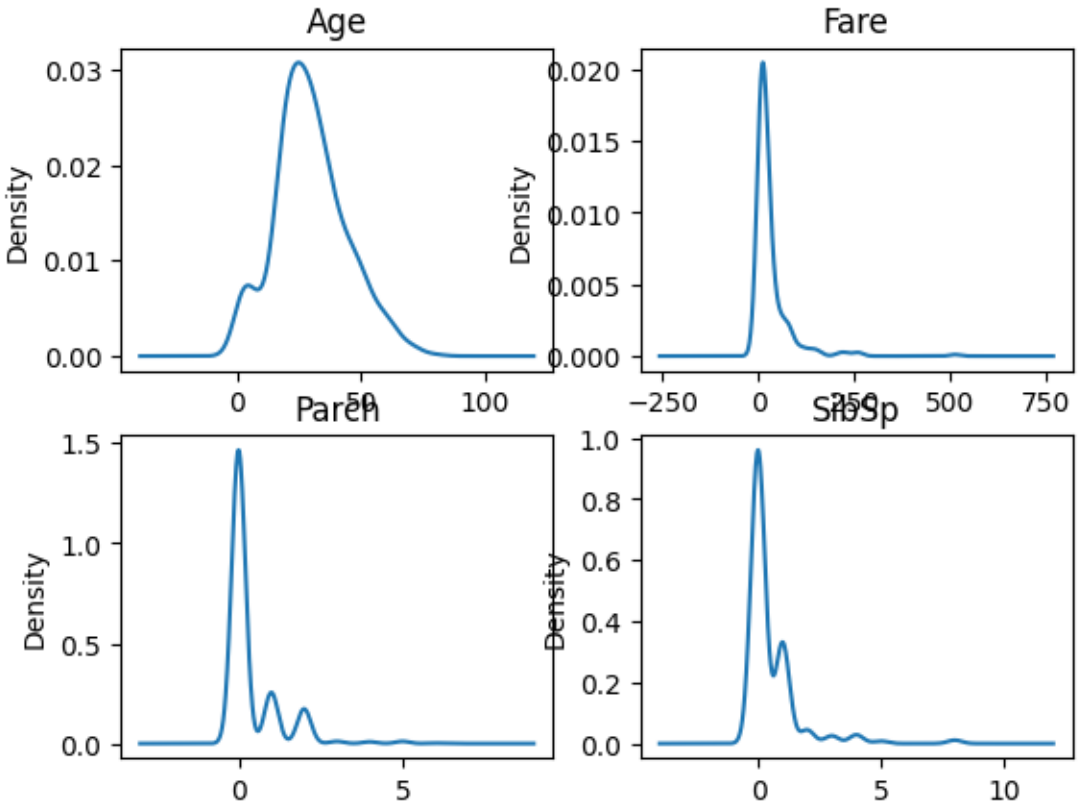
Documentation - Pandas - DataFrame.plot()

[82]:
```
continuous_numeric_features = ['Age', 'Fare', 'Parch', 'SibSp']

# Insert your code here
fig, axes = plt.subplots(nrows=2, ncols=2)
```

```
data['Age'].plot(kind='density', ax=axes[0,0], title='Age')
data['Fare'].plot(kind='density', ax=axes[0,1], title='Fare')
data['Parch'].plot(kind='density', ax=axes[1,0], title='Parch')
data['SibSp'].plot(kind='density', ax=axes[1,1], title='SibSp')
```

[82]: <Axes: title={'center': 'SibSp'}, ylabel='Density'>



Comments : 'Fare', 'Parch', and 'SibSp' have a distribution close to normal, but with a left side skew, "Age" have a distribution a a bit different from the other but maybe it's close enough to Gaussian.

1.4 Model Training, Selection, and Evaluation

We will now fit our Naive Bayes model to data, compare variations on the model, and evaluate our best model.

1.4.1 e. Initial Model Fit

We will fit a Naive Bayes models with Normally distributed class-conditional distributions on the features. In the cell below, define a GaussianNB object and use the fit() function to fit to training data. Use the predict() function to predict labels for the test data. Compute and report the accuracy of your predictions.

Documentation - Scikit-Learn - naive_bayes.GaussianNB

[83]:
```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(X_train, Y_train)#insert your code to fit the training Data
Y_pred = nb.predict(X_test)#insert your code here

# Evaluate Accuracy
# Insert your code here
from sklearn import metrics
print("Accuracy score:")
metrics.accuracy_score(Y_test, Y_pred)
```

```
Accuracy score:
```

[83]:
```
0.7847533632286996
```

Using the attributes of the GaussianNB class, display the means of the class-conditional distributions for each label category (Died, Survived). The theta_ attribute of your classifier will return a 2xM array where each row contains means for each of the M features. Print the output in the following format:

```
feature-name (label-category-name): value
```

For example:

```
Age (Died): 33.732530
```

[84]:
```
from sklearn.naive_bayes import GaussianNB
features = ['Age', 'Embarked', 'Fare', 'Parch', 'Pclass', 'Sex', 'SibSp']
# Insert your code here
for i in range(7):
    print(features[i] + "(Died):")
    print(nb.theta_[0][i])
for i in range(7):
    print(features[i] + "(Survived):")
    print(nb.theta_[1][i])
```

```
Age(Died):
29.954216867469878
Embarked(Died):
1.6602409638554216
Fare(Died):
22.010219277108433
Parch(Died):
0.3253012048192771
Pclass(Died):
2.539759036144578
Sex(Died):
0.8578313253012049
```

```
SibSp(Died):
0.5927710843373494
Age(Survived):
28.327430830039525
Embarked(Survived):
1.4031620553359683
Fare(Survived):
48.86006324110672
Parch(Survived):
0.4505928853754941
Pclass(Survived):
1.9960474308300395
Sex(Survived):
0.32806324110671936
SibSp(Survived):
0.4901185770750988
```

1.4.2 f. Model Selection

We will use cross-validation to choose among a number of models, and select the best to evaluate on test data. To begin, use the Scikit-Learn cross_val_score() function to perform 10-fold cross validation of the GaussianNB classifier. **Print the mean and standard deviation of the cross-validation score.**

Documentation - Scikit-Learn - model_selection.cross_val_score

[85]:
```
from sklearn.model_selection import cross_val_score
scoring = 'accuracy' # use Accuracy scoring method in cross-validation
cv=10 # 10-fold cross validation

# Cross-Validation on Baseline model
# Insert your code here
cv_score = cross_val_score(nb, X_train, Y_train, cv=cv, scoring=scoring)
import numpy as np
print("Mean CV score:")
print(np.mean(cv_score))
print("STD CV score:")
print(np.std(cv_score))
```

```
Mean CV score:
0.7963591135232926
STD CV score:
0.06258843429325842
```

How predictive is each feature? For each feature in the data compute the cross validation score using *only* that feature as input. Report the mean and standard deviation of the CV score for each feature. Report which feature is most predictive.

Note: cross_val_score() expects a 2D array as input features, so if you simply pass in X['feature'] it will complain. You'll need to temporarily copy and augment the feature. I've included some

commented code as an example.

[86]:
```
## You may need the following code to pass single features to cross_val_score
# Xtmp = X_train[feat].values
# Xtmp = Xtmp.reshape((len(X_train[feat]), 1))

# Train / predict using each feature
# Insert your code here
accuracy = []
for f in features:
    Xtmp = X_train[f].values
    Xtmp = Xtmp.reshape((len(X_train[f]), 1))
    nb_tmp = GaussianNB()
    nb_tmp.fit(Xtmp, Y_train)
    cv_score = cross_val_score(nb_tmp, Xtmp, Y_train, cv=cv, scoring=scoring)
    print(f + ":")
    print("Mean CV score:")
    print(np.mean(cv_score))
    print("STD CV score:")
    print(np.std(cv_score))
    print()
    accuracy.append(np.mean(cv_score))
print( features[accuracy.index(max(accuracy))] + ' is most predictive.')
```

```
Age:
Mean CV score:
0.6376978742650385
STD CV score:
0.019115743379569502

Embarked:
Mean CV score:
0.6407055630936228
STD CV score:
0.034829512223924386

Fare:
Mean CV score:
0.6706919945725917
STD CV score:
0.033021969436852754

Parch:
Mean CV score:
0.6212573496155586
STD CV score:
0.006462915712116252
```

```
Pclass:
Mean CV score:
0.6720940750791498
STD CV score:
0.029426803303276555

Sex:
Mean CV score:
0.7875169606512891
STD CV score:
0.07418224800703502

SibSp:
Mean CV score:
0.5386928991406603
STD CV score:
0.12847161433989057

Sex is most predictive.
```

Drop highly correlated features. Recall that Naive Bayes models features as conditionally independent. However, we found that the Pearson correlation coefficient between 'Pclass' and 'Fare' indicates that the two features are highly correlated. Create a temporary copy of the Training and Test data (e.g. X_train.copy()) and **drop the Fare feature** but keep the 'Pclass' feature. You may do this using the DataFrame.drop() function.

Report the mean/stdev of the new CV score without this feature.

Documentation - Pandas - DataFrame.drop

[95]:
```
# Drop Fare
# Insert your code here
x_train_copy = X_train.copy()
x_test_copy = X_test.copy()

x_train_copy.drop(columns=['Fare'])
x_test_copy.drop(columns=['Fare'])

nb_copy = GaussianNB()
nb_copy.fit(x_train_copy, Y_train)
cv_score = cross_val_score(nb, x_train_copy, Y_train, cv=cv, scoring=scoring)
print("Mean CV score:")
print(np.mean(cv_score))
print("STD CV score:")
print(np.std(cv_score))
```

```
Mean CV score:
0.7963591135232926
STD CV score:
0.06258843429325842
```

Now repeat the above procedure, again copying the original training data, and **drop the 'Pclass' feature** while keeping the 'Fare' feature. Again report the mean/stdev of the CV score of the classifier without this feature.

[96]:
```
# Drop Pclass (highly-correlated with Fare)
# Insert your code here
x_train_copy = X_train.copy()
x_test_copy = X_test.copy()

x_train_copy.drop(columns=['Pclass'])
x_test_copy.drop(columns=['Pclass'])

nb_copy = GaussianNB()
nb_copy.fit(x_train_copy, Y_train)
cv_score = cross_val_score(nb, x_train_copy, Y_train, cv=cv, scoring=scoring)
print("Mean CV score:")
print(np.mean(cv_score))
print("STD CV score:")
print(np.std(cv_score))
```

```
Mean CV score:
0.7963591135232926
STD CV score:
0.06258843429325842
```

1.5 Testing the Model

If you have done things properly you should see that discarding the 'Pclass' feature, but keeping 'Fare', leads to the best prediction accuracy. In the code below we will select this best performing model and evaluate it on the test data.

1.5.1 g. Select the model and test

In the cell below perform the following steps: * Permanently drop 'Pclass' feature from training and test data * Train a GaussianNB() model on the modified training data using the fit() function * Evaluate the model on Test data using the predict() function * Report prediction accuracy

[98]:
```
#Drop Pclass from both test and train data with axis = 1, inplace = True
X_test.drop(columns=['Pclass'])
X_train.drop(columns=['Pclass'])
nb_new = GaussianNB()
nb_new.fit(X_train, Y_train)
Y_pred = nb_new.predict(X_test)
```

[99]:
```
from sklearn.metrics import accuracy_score

# Insert your code here
accuracy_score(Y_test, Y_pred)
```

[99]: 0.7847533632286996

1.5.2 h. Evaluation metrics

Now we will look beyond accuracy to evaluate our classifier. One useful statistic for evaluating classifiers is the *confusion matrix*, which enumerates categories of correct and incorrect classifications. For more information see the Wikipedia article on the confusion matrix. Compute the confusion matrix and report whether one class is confused for another class more often, or whether they are about the same (within a couple of points).

Documentation - Scikit-Learn - metrics.confusion_matrix

[103]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_train, nb_new.predict(X_train))
print(cm)
print('Your answer here')
```

```
[[350  65]
 [ 71 182]]
Your answer here
```

[106]:
```
from sklearn.metrics import recall_score, precision_score
print(metrics.precision_score(Y_test, nb_new.predict(X_test)))
print(metrics.recall_score(Y_test, nb_new.predict(X_test)))
# Insert your code
```

```
0.711340206185567
0.7752808988764045
```

2 Conclusion

Congratulations, you have succesfully explored a dataset, processed it for classification, trained a Naive Bayes Classifier and evaluated it's performance!!