**Data mining class** Sayeh Sobhani

Second assignment student no. 99422102

## 1.

The first dataset of this assignment is about apartment rental offers in Germany. The goal is to predict the living space based on different features with a linear model.

Let’s first review several cross-validation approaches:

**K-Fold** cross-validation: Most common

**Leave One Out (LOO)**: Takes each data point as the 'test sample' once, and trains the model on the rest n-1 data points. Thus, it trains n total models.

Advantage: Utilizes the data well since each model is trained on n-1 samples

Disadvantage: Computationally expensive

**Leave P-Out (LPO)**: Create all possible splits after leaving p samples out. For n data points, there are (nCp) possible train-test splits.

(**For classification problems**) **Stratified K-Fold**: Ensures that the relative class proportion is approximately preserved in each train and validation fold. Important when there is huge class imbalance (e.g., 98% good customers, 2% bad).

## 2.

### 2.1) Compare LASSO vs. RIDGE regression and explain each in a paragraph.

**a:**

The acronym “LASSO” stands for Least Absolute Shrinkage
and Selection Operator.

LASSO regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models (i.e., models with fewer parameters). This particular type of regression is well-suited for models showing high levels of multicollinearity or when you want to automate certain parts of model selection, like variable selection/parameter elimination.

Ridge regression is a way to create a parsimonious model when the number of predictor variables in a set exceeds the number of observations, or when a data set has multicollinearity (correlations between predictor variables).

Tikhonov’s method is basically the same as ridge regression, except that Tikhonov’s has a larger set. It can produce solutions even when your data set contains a lot of statistical noise (unexplained variation in a sample).

Ridge regression decreases the complexity of a model but does not reduce the number of variables since it never leads to a coefficient been zero rather only minimizes it. Hence, this model is not good for feature reduction.

Lasso sometimes struggles with some types of data. If the number of predictors (p) is greater than the number of observations (n), Lasso will pick at most n predictors as non-zero, even if all predictors are relevant (or may be used in the test set).

If there are two or more highly collinear variables then LASSO regression select one of them randomly which is not good for the interpretation of data

Let's think of an example where we have a large dataset, let's say it has 10,000 features. And we know that some of the independent features are correlated with other independent features. Then think, which regression would you use, RIDGE or LASSO?

Let's discuss it one by one. If we apply RIDGE regression to it, it will retain all of the features but will shrink the coefficients. But the problem is that model will still remain complex as there are 10,000 features, thus may lead to poor model performance.

Instead of RIDGE what if we apply LASSO regression to this problem. The main problem with LASSO regression is when we have correlated variables, it retains only one variable and sets other correlated variables to zero. That will possibly lead to some loss of information resulting in lower accuracy in our model.

Then what is the solution for this problem? Actually, we have another type of regression, known as Elastic Net regression, which is basically a hybrid of ridge and lasso regression.

**2.2)**

**a:**

**2.3) What's the outcome of increasing folds?**

**a:**

With increasing k, two things happen:

- Your k−1 training folds increase in size

- Your validation folds decrease in size

From the first point you can draw the conclusion that your kk models become more similar since your training data becomes more similar since you split off less data for the validation sets in the k-th fold. Which might lead to less between model-variance. Moreover, since you are training on more data, the relative model complexity (i.e., compared to your data) decreases. In the bias-variance-trade-off graph, this means we are moving towards the left, i.e., we trade less model variance for more model bias (please note that model variance here has a more general and conceptual meaning than the calculated variance or standard deviation between folds). The reason is that we are fitting a model with constant complexity to more data, as k increases, i.e., it becomes harder to learn the training data.

However, we are not only increasing the size of our training set with increasing kk. We are also decreasing the validation set in size (see point two from above). Therefore, there might be higher between-fold variance with regards to our validation sets. This might be more relevant if the overall dataset carries higher variance or more outliers.

**2.4) What's Leave One Out (LOOCV) and where is it used?**

**a:**

Leave-one-out cross-validation is a special case of cross-validation where the number of folds equals the number of instances in the data set. Thus, the learning algorithm is applied once for each instance, using all other instances as a training set and using the selected instance as a single-item test set. This process is closely related to the statistical method of jack-knife estimation.

LOOCV, procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model.

**2.6) Explain what 5x2 cross validation is and where it is used in a paragraph.**

**a**:

5x2 CV refer to a 5 repetition of a 2-fold. If you do a 2-fold (50/50 split between train and test), repeat it 4 more times. The 5x2cv was popularized by Dietterich as a way of obtaining not only a good estimate of the generalization error but also a good estimate of the variance of that error (in order to perform statistical tests).

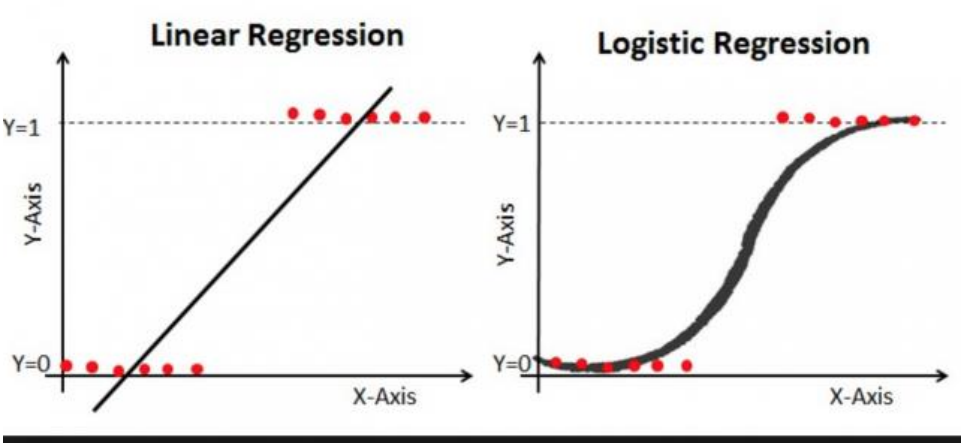
This is mainly used for better estimations of model performance, like running statistical tests on whether one model performs statistically-significantly better than another.

## 3.Coding problem

In this section we are going to use a mobile price classification dataset from Kaggle to answer some questions.

First, let's talk a little about logistic regression.

In statistics, the logistic model is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick.




I imported the necessary packages for logistic regression and started the process.

The target and features are separated and 80 percent of the data is split to make a training set.

3.1.The price range has 4 classes. A logistic regression model is learned and the precision, recall and f1-score for this part is as bellow:

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.80 | 0.79 | 0.80 |
| 1 | 0.54 | 0.58 | 0.56 |
| 2 | 0.52 | 0.36 | 0.43 |
| 3 | 0.58 | 0.81 | 0.68 |
| accuracy | | | 0.61 |
| macro avg | 0.61 | 0.63 | 0.61 |
| weighted avg | 0.61 | 0.61 | 0.60 |

The results if we normalize data:

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.96 | 0.98 | 0.97 |
| 1 | 0.92 | 0.95 | 0.93 |
| 2 | 0.98 | 0.92 | 0.95 |
| 3 | 0.98 | 0.99 | 0.98 |
| accuracy | | | 0.96 |
| macro avg | 0.96 | 0.96 | 0.96 |
| weighted avg | 0.96 | 0.96 | 0.96 |

3.2.In the test section of the dataset, most of the data are in class 2:

```
In [13]: y_test.value_counts()
Out[13]:
2    113
0    100
1    100
3     87
Name: price_range, dtype: int64
```

As for the train part, most of the data are from 3rd class:

```
In [14]: y_train.value_counts()
Out[14]:
3    413
0    400
1    400
2    387
Name: price_range, dtype: int64
```

3.3.I then merged the non-zero classes in the price range column into one class to make the 4-class target a 2-class target. This is how the test and train data looks like:

```
In [21]: y_train.value_counts()
Out[21]:
1    1200
0     400
Name: price_range, dtype: int64
```

```
In [22]: y_test.value_counts()
Out[22]:
1    300
0    100
Name: price_range, dtype: int64
```

3.4.I then run the linear regression model for the two-class problem and the figure below shows the results:

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.92 | 0.85 | 0.89 |
| 1 | 0.96 | 0.98 | 0.97 |
| accuracy | | | 0.95 |
| macro avg | 0.94 | 0.91 | 0.93 |
| weighted avg | 0.95 | 0.95 | 0.95 |

3.5. Training a machine learning model on an imbalanced dataset can introduce unique challenges to the learning problem. Imbalanced data typically refers to a classification problem where the number of observations per class is not equally distributed; often you'll have a large number of data/observations for one class (referred to as the majority class), and much fewer observations for one or more other classes (referred to as the minority classes). For example, suppose you're building a classifier to classify a credit card transaction a fraudulent or authentic - you'll likely have 10,000 authentic transactions for every 1 fraudulent transaction, that's quite an imbalance!

To take care of this problem we ca take several approaches:

**Class weight**:

One of the simplest ways to address the class imbalance is to simply provide a weight for each class which places more emphasis on the minority classes such that the end result is a classifier which can learn equally from all classes.

To calculate the proper weights for each class, you can use the sklearn utility function

```
from sklearn.utils.class_weight import compute_class_weight
weights = compute_class_weight('balanced', classes, y)
```

**Oversampling**:

Another approach towards dealing with a class imbalance is to simply alter the dataset to remove such an imbalance. In this section, I'll discuss common techniques for oversampling the minority classes to increase the number of minority observations until we've reached a balanced dataset.

*Random oversampling*

*SMOTE*

*ADASYN*

**Undersampling**:

Rather than oversampling the minority classes, it's also possible to achieve class balance by undersampling the majority class - essentially throwing away data to make it easier to learn characteristics about the minority classes.

*Random undersampling*

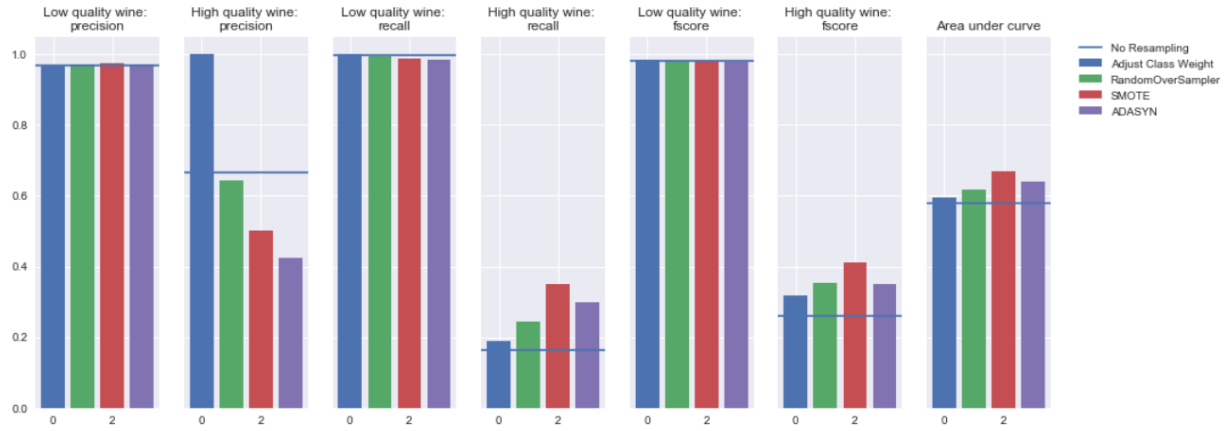
*Near miss (1,2)*

*Tomeks links*

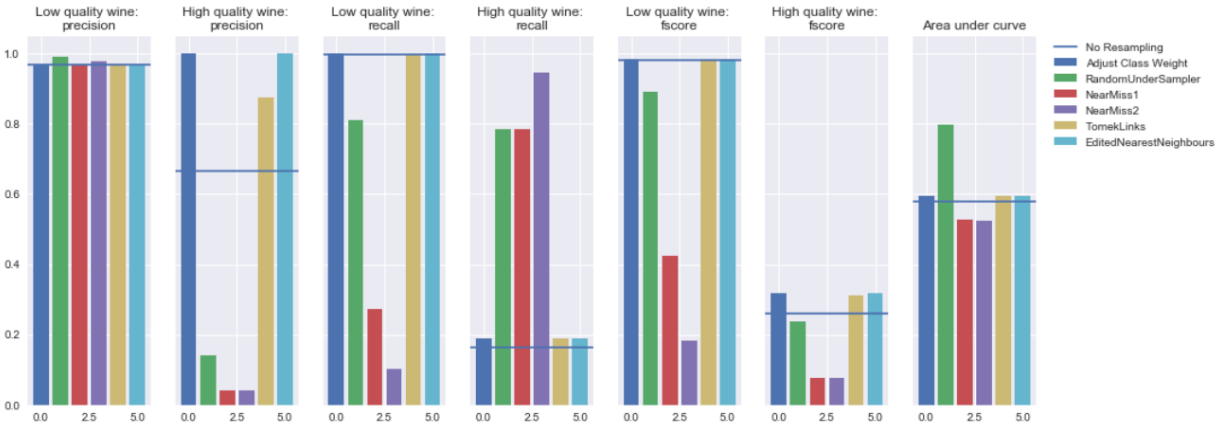
*Edited nearest neighbors*

Let’s take a look at each strategy for logistic regression:

## Oversampling




## Undersampling




I used SMOTE form the oversampling strategy to balance the data in two classes and then fit the regression and the results are as we can see here:

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.92 | 0.94 | 0.93 |
| 1 | 0.94 | 0.92 | 0.93 |
| accuracy | | | 0.93 |
| macro avg | 0.93 | 0.93 | 0.93 |
| weighted avg | 0.93 | 0.93 | 0.93 |

3.6. Let’s review forward selection:

In forward selection, we start with a null model and then start fitting the model with each individual feature one at a time and select the feature with the minimum p-value. Now fit a model with two features by trying combinations of the earlier selected feature with all other remaining features. Again, select the feature with the minimum p-value. Now fit a model with three features by trying combinations of two previously selected features with other remaining features. Repeat this process until we have a set of selected features with a p-value of individual features less than the significance level.

As for the Area under curve:

AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease.

By forward selection, I picked several features with higher AUC to run the regression model on. The picked features are: ['mobile_wt', 'fc', 'pc', 'battery_power', 'px_width', 'touch_screen']

3.7. And the results of logistic regression for these features picked by forward selection are:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.32 | 0.49 | 0.39 | 93 |
| 1 | 0.33 | 0.09 | 0.14 | 102 |
| 2 | 0.26 | 0.10 | 0.14 | 102 |
| 3 | 0.32 | 0.58 | 0.41 | 103 |
| accuracy | | | 0.31 | 400 |
| macro avg | 0.31 | 0.32 | 0.27 | 400 |
| weighted avg | 0.31 | 0.31 | 0.27 | 400 |

3.8. In the previous part, 6 features were picked by forward selection. In this section, 6 features are to be picked with PCA (pca = PCA(n_components=6)). Principal component analysis (PCA) is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss. It does so by creating new uncorrelated variables that successively maximize variance.

3.9. In this part 6 features are picked by PCA and a logistic regression is run. The results are:

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.50 | 1.00 | 0.67 |
| 1 | 1.00 | 0.09 | 0.16 |
| 2 | 1.00 | 0.22 | 0.35 |
| 3 | 0.56 | 1.00 | 0.72 |
| accuracy | | | 0.57 |
| macro avg | 0.77 | 0.58 | 0.48 |
| weighted avg | 0.77 | 0.57 | 0.47 |

3.10. for this step, I tried eliminating features with backward selection.

These are the features picked by this method: 'four_g', 'blue', 'int_memory', 'wifi'

Then the logistic regression model was run with these selected features and we can see a decrease in all three scores for this model, comparing to the model run in part 7 for forward selection:

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.20 | 0.23 | 0.21 |
| 1 | 0.23 | 0.09 | 0.13 |
| 2 | 0.22 | 0.29 | 0.25 |
| 3 | 0.29 | 0.34 | 0.31 |
| accuracy | | | 0.24 |
| macro avg | 0.23 | 0.24 | 0.23 |
| weighted avg | 0.24 | 0.24 | 0.23 |

## 4.

### 4.1) Can logistic regression be used for multi-class classification, how? Explain.

**a:**

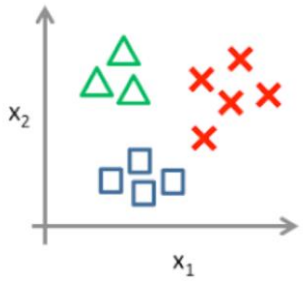
To solve this problem, a method called One vs. All is applied.

To explain the One vs. all (or in some texts known as One vs. Rest) method:

In one-vs-All classification, for the N-class instances dataset, we have to generate the N-binary classifier models. The number of class labels present in the dataset and the number of generated binary classifiers must be the same. In other words, we have to generate the same number of classifiers as the class labels are present in the dataset.

For example, if we have three classes, we create three classifiers (shown in the figures):

Let red, green and blue be three classes.

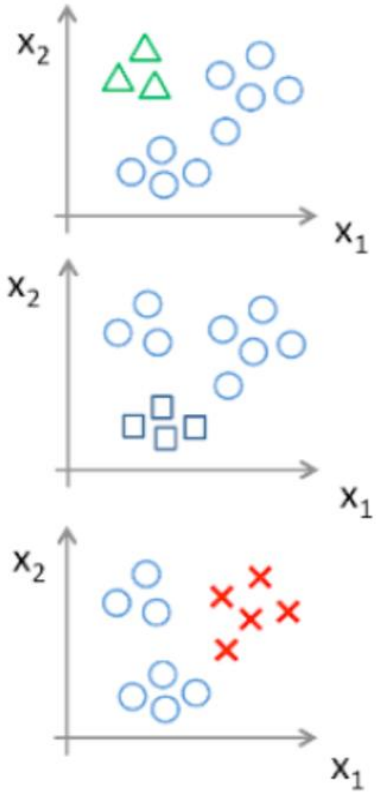



We will have three classifiers:

Classifier 1: [Green] vs [Red, Blue]

Classifier 2: [Blue] vs [Green, Red]

Classifier 3: [Red] vs [Blue, Green]



Now, after creating a training dataset for each classifier, we provide it to our classifier model and train the model by applying an algorithm. Then By analyzing the probability scores, we predict the result as the class index having a maximum probability score.

**4.4) Name two other feature selection methods (except backward and forward selection) and explain each in one paragraph.**

**a:**

*1.Recursive Feature elimination:*

It is a greedy optimization algorithm which aims to find the best performing feature subset. It repeatedly creates models and keeps aside the best or the worst performing feature at each iteration. It constructs the next model with the left features until all the features are exhausted. It then ranks the features based on the order of their elimination.

*2. Exhaustive Feature Selection*

In exhaustive feature selection, the performance of a machine learning algorithm is evaluated against all possible combinations of the features in the dataset. The feature subset that yields best performance is selected. The exhaustive search algorithm is the most greedy algorithm of all the wrapper methods since it tries all the combination of features and selects the best. A downside to exhaustive feature selection is that it can be slower compared to step forward and step backward method since it evaluates all feature combinations.

**4.5) What are some disadvantages of forward and backward selection? Explain.**

**a:**

We know that while a set of variables can have significant predictive ability, a particular subset of them may not. Unfortunately, forward selection does not have the capacity to identify less predictive individual variables that may not enter the model to demonstrate their joint behavior.

However, backward selection has the advantage to assess the joint predictive ability of variables as the process starts with all variables being included in the model. Backward selection also removes the least important variables early on and leaves only the most important variables in the model.

One disadvantage of the backward selection method is that once a variable is eliminated from the model it is not re-entered again. However, a dropped variable may become significant later in the final model.

**4.6) Explain Linear Discriminant Analysis (LDA) and compare LDA to PCA for solving a problem.**

**a**:

Linear discriminant analysis (LDA) is generally used to classify patterns between two classes; however, it can be extended to classify multiple patterns.

LDA assumes that all classes are linearly separable and according to this multiple linear discrimination function representing several hyperplanes in the features space are created to distinguish the classes. If there are two classes then the LDA draws one hyperplane and projects the data onto this hyperplane in such a way as to maximize the separation of the two categories. This hyperplane is created according to the two criteria considered simultaneously:

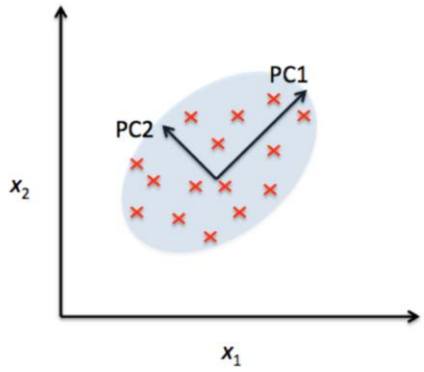
Maximizing the distance between the means of two classes;

Minimizing the variation between each category.

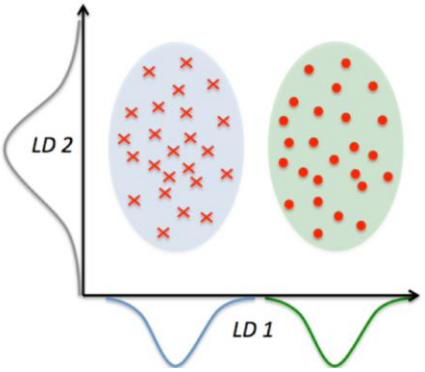
It provides accepted accuracy and is widely used in BCI (Brain–computer interface) systems.

Now to compare LDA vs. PCA:

Both LDA and PCA are linear transformation techniques: LDA is a supervised whereas PCA is unsupervised – PCA ignores class labels. We can picture PCA as a technique that finds the directions of maximal variance:




In contrast to PCA, LDA attempts to find a feature subspace that maximizes class separability (note that LD 2 would be a very bad linear discriminant in the figure above). LDA makes assumptions about normally distributed classes and equal class covariances whereas PCA tends to result in better classification results in an image recognition task if the number of samples for a given class was relatively small.



## 4.7) How can we use Statistical Significance Tests to Compare Models?

**a:**

Statistical significance tests are designed to address whether the difference between mean skill scores is real or the result of a statistical fluke and quantify the likelihood of the samples of skill scores being observed given the assumption that they were drawn from the same distribution. If this assumption, or null hypothesis, is rejected, it suggests that the difference in skill scores is statistically significant.

A big part of applied machine learning is **model selection**. To address this, we choose the model with the best skill.

The model whose estimated skill when making predictions on unseen data is best. This might be maximum accuracy or minimum error in the case of classification and regression problems respectively.

To determine how much we can trust the estimated skill of each model, we can use **statistical hypothesis** testing to address this.

The assumption of a statistical test is called the null hypothesis and we can calculate statistical measures and interpret them in order to decide whether or not to accept or reject the null hypothesis.

In the case of selecting models based on their estimated skill, we are interested to know whether there is a real or statistically significant difference between the two models.

Given that using statistical hypothesis tests seems desirable as part of model selection, choosing a proper Hypothesis test that is suitable for our specific use is important. It is common practice to evaluate classification methods using classification accuracy, to evaluate each model using 10-fold cross-validation, to assume a Gaussian distribution for the sample of 10 model skill estimates, and to use the mean of the sample as a summary of the model's skill.

Four types of tests of significance in statistics are: 1. Student's T-Test or T-Test 2. F-test or Variance Ratio Test 3. Fisher's Z-Test or Z-Test 4. $X_2$-Test (Chi-Square Test).

**4.8) What is Matthews Correlation Coefficient (MCC) and where is it used?**

**a**:

The Matthews correlation coefficient (MCC) or phi coefficient is used in machine learning as a measure of the quality of binary (two-class) classifications.

Similar to F1 score, MCC is a single-value metric that summarizes the confusion matrix. A confusion matrix, also known as an error matrix, has four entries: True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN). The main benefit gained by using MCC instead of F1 score can be guessed easily by peeking into their formula:

$$F1 = \frac{2TP}{2TP + FN + FP}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TN + FN)(FP + TP)(TN + FP)(FN + TP)}}$$

F1 score ignores the count of True Negatives. In contrast, MCC kindly extends its care to all four entries of the confusion matrix. Davide Chicco, the author of Ten quick tips for machine learning in computational biology, commented that MCC "is high only if your classifier is doing well on both the negative and the positive elements."

In times when precision and recall are important for us, F1 score, in this case, merges precision and recall in a more interpretable way than MCC does.

Only if the cost of low precision and low recall is really unknown or unquantifiable, MCC is preferred over F1 score as it is a more 'balanced' assessment of classifiers, no matter which class is positive.

# References


1. jeremyjordan.me

2. towardsdatascience.com

3. Wikipedia

4. biologydiscussion.com

5. machinelearningmastery.com

6. Chapter 2 of Brain–computer interfaces and their applications, An Industrial IoT Approach for Pharmaceutical Industry Growth, Academic Press, 2020, Pages 31-54

7. sebastianraschka.com

8. M. Martinez and A. C. Kak. PCA versus LDA. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 23(2):228–233, 2001))

9. Chowdhury MZ, Turin TC. Variable selection strategies and its importance in clinical prediction modelling. Family medicine and community health. 2020;8(1)

10. Stackabuse

11. paper of family medicine and community health: Variable selection strategies and its importance in clinical prediction modelling

12. Andrew Ng's video about logistic regression

13. Dietterich TG. Approximate statistical tests for comparing supervised classification learning algorithms. Neural computation. 1998 Oct 1;10(7):1895-923

14. Sammut C, Webb GI, editors. Encyclopedia of machine learning. Springer Science & Business Media; 2011 Mar 28

15. statisticshowto.com

16. geeksforgeeks.org

17. analyticsvidhya.com

18. The Elements of Statistical Learning" by Hastie et al

19. datascience.stackexchange.com

20. scikit-learn.org

21. kdnuggets.com

22. pythonhealthcare.org

23. askpython.com

24. pandas.pydata.org

# Appendix

Code for part 1:

```
#%%importing data and packages

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

from sklearn.metrics import mean_squared_error as mse
from sklearn.linear_model import LinearRegression, Ridge, Lasso, LogisticR
egression
from sklearn.metrics import classification_report
from sklearn.utils import shuffle
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
from sklearn.decomposition import PCA
from sklearn.datasets import make_classification
```

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn import preprocessing
from numpy import mean
from numpy import std
from sklearn.model_selection import train_test_split, cross_val_predict
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_recall_fscore_support as F1


df1 = pd.read_csv (r'C:\Users\Asus\Desktop\Data Mining\02_assignment\data\
part1\immo_data.csv')
df2 = pd.read_csv (r'C:\Users\Asus\Desktop\Data Mining\02_assignment\data\
part3\train.csv')

#%%Understanding data for first dataset

#to get a better understanding of how our data looks
df1.shape
df1.info()
df1.describe()
df1.head(10)
df1.columns
#duplicated datapoints
df1.duplicated(keep=False).any()
#the mean for each numeric feature?
df1._get_numeric_data().mean()
#useless columns deletion '',
df1 = df1.drop(columns=['description'])
df1 = df1.drop(columns=['facilities', 'telekomHybridUploadSpeed'])
df1 = df1.drop(columns=['scoutId'])
df1 = df1.drop(columns=['geo_bln','date'])
df1 = df1.drop(columns=['houseNumber','livingSpaceRange','firingTypes'])
df1 = df1.drop(columns=['streetPlain','street','geo_krs','regio2','regio3'
])
df1 = df1.drop(df1[df1['livingSpace'] == 0.0].index)
df1 = df1.drop(df1[df1['totalRent'] == 0.0].index)

#how null data looks like
df1.isna()

df1.columns

#columns with null data
df1.isna().sum()/len(df1)
```

```
#which columns hve >50 percent null
df1.columns[((df1.isna().sum()/len(df1)) > 0.50)]

#removing those columns with >50 null
df1 = df1.drop(columns=df1.columns[((df1.isna().sum()/len(df1)) > 0.50)])
df1.columns
df1.shape

#filling the remaining data with means and max seen data
    #numeric
df1._get_numeric_data().mean()
df1.fillna(df1._get_numeric_data().mean(),inplace = True)
df1.isna().sum()
    #Categorical most seens
for cols in df1.columns:
    if df1[cols].dtype == 'object' or df1[cols].dtype == 'bool':
        print('column : ',cols)
        print(df1[cols].value_counts().head())

     #Fill categorical nan with most seen data in each feature
for cols in df1.columns:
    if df1[cols].dtype == 'object' or df1[cols].dtype == 'bool':
        print('cols : {} , value : {}'.format(cols , df1[cols].value_count
s().head(1).index[0]))
        df1[cols].fillna(df1[cols].value_counts().head(1).index[0],inplace
 = True)

#to check if nan left
df1.isna().sum()
df1.shape
df1.info()
#no null data left

#outliers finding and removing
for cols in df1.columns:
    if df1[cols].dtype == 'int64' or df1[cols].dtype == 'float64':
        upper_range = df1[cols].mean() + 3 * df1[cols].std()
        lower_range = df1[cols].mean() - 3 * df1[cols].std()

        indexs = df1[(df1[cols] > upper_range) | (df1[cols] < lower_range)
].index
        df1 = df1.drop(indexs)

df1.info()
```

```
df1.shape

#%%categorical feature inspection
#the number of categories in each feature
for cols in df1.columns:
    if df1[cols].dtype == 'object' or df1[cols].dtype == 'bool':
        print('cols : {} , unique values : {}'.format(cols,df1[cols].nuniq
ue()))


#the categories and their count in each feature
for cols in df1.columns:
    if df1[cols].dtype == 'object' or df1[cols].dtype == 'bool':
        print('cols : {} ,\n {}'.format(cols,df1[cols].value_counts()))

#regio1 edition based on plots from base rent amounts in line 75
df1['regio1'].value_counts()
def edit_regio1(x):
    if x in ['Hamburg','Bremen','Saarland']:
        return 'other'
    else:
        return x
df1['regio1_'] = df1['regio1'].apply(edit_regio1)
df1 = df1.drop(columns = ['regio1'])
df1['regio1_'].value_counts()*100 / len(df1)

    #visualization
#plt_regio1_= df1['regio1_'].value_counts().plot(kind='bar')

#heatingType
df1['heatingType'].value_counts()
others = list(df1['heatingType'].value_counts().tail(12).index)
def edit_heatingType(x):
    if x in others:
        return 'other'
    else:
        return x

df1['heatingType_'] = df1['heatingType'].apply(edit_heatingType)
df1 = df1.drop(columns = ['heatingType'])
df1['heatingType_'].value_counts()*100 / len(df1)

#telekomTvOffer
df1['telekomTvOffer'].value_counts()
others = list(df1['telekomTvOffer'].value_counts().tail(2).index)
```

```
def edit_telekomTvOffer(x):
    if x in others:
        return 'other'
    else:
        return x

df1['telekomTvOffer_'] = df1['telekomTvOffer'].apply(edit_telekomTvOffer)
df1 = df1.drop(columns = ['telekomTvOffer'])
df1['telekomTvOffer_'].value_counts()*100 / len(df1)

#typeOfFlat
df1['typeOfFlat'].value_counts()
others = list(df1['typeOfFlat'].value_counts().tail(6).index)
def edit_typeOfFlat(x):
    if x in others:
        return 'other'
    else:
        return x

df1['typeOfFlat_'] = df1['typeOfFlat'].apply(edit_typeOfFlat)
df1 = df1.drop(columns = ['typeOfFlat'])
df1['typeOfFlat_'].value_counts()*100 / len(df1)

#condition
df1['condition'].value_counts()
others = list(df1['condition'].value_counts().tail(4).index)
def edit_condition(x):
    if x in others:
        return 'other'
    else:
        return x

df1['condition_'] = df1['condition'].apply(edit_condition)
df1 = df1.drop(columns = ['condition'])
df1['condition_'].value_counts()*100 / len(df1)

#%%normalization of numeric data
for cols in df1.columns:
    if df1[cols].dtype == 'int64' or df1[cols].dtype == 'float64':
         if cols != 'livingSpace':
              df1[cols] = ((df1[cols] - df1[cols].mean())/(df1[cols].std()
))

#%%correlation
```

```
df1.corr().livingSpace.sort_values()

#%%Dummy variables for categorical
columns = []
for cols in df1.columns:
    if df1[cols].dtype == 'object' or df1[cols].dtype == 'bool':
        columns.append(cols)
columns

dummies_feature = pd.get_dummies(df1[columns])
dummies_feature.head()
dummies_feature.shape
df1 = pd.concat([df1, dummies_feature], axis=1)
df1.head()
df1 = df1.drop(columns=columns)
df1.head()
df1.info()

#%%shuffling data and dividing target and feature, test and train

#most correlated feature(noRooms) and living space as target
df1 = shuffle(df1)
y = df1['livingSpace'].values
x = df1['noRooms'].values

print(x.shape)
print(y.shape)

#splitting test and train
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, r
andom_state=0)

#print(x_train.shape)
#print(y_train.shape)

#print(x_test.shape)
#print(y_test.shape)

#%%linear regr no package and accuracy function
#y^=w*x+b

def LinearRegr_grad(x, y, lr=0.001, epochs=500):
    w = 0
    b = 0
    n = np.float(x.shape[0])
```

```
    losses = []

    for i in range(epochs):
        if i % 100 == 0:
            print(i)

        y_hat = w * x + b

        mse = (1/n) * np.sum((y - y_hat)**2)
        losses.append(mse)

        dw = (-2/n) * np.sum(np.dot(x.T, (y - y_hat)))
        db = (-2/n) * np.sum(y - y_hat)

        w = w - lr * dw
        b = b - lr * db

    return w, b


def accur(actual, predicted):
    truelb = 0
    for i in range(len(actual)):
        if predicted[i] >= actual[i]*0.9 and predicted[i] <= actual[i]*1.1
:
            truelb += 1
    return truelb / float(len(actual)) * 100.0


#%% k fold (5,10)
# 5fold cv
kf = KFold(n_splits=5, random_state=None)

acc_score = []
errors = []

for train_index , test_index in kf.split(x):



    w, b = LinearRegr_grad(x_train, y_train, epochs=2000)
    y_hat = w * x_test + b

    acc_score.append(accur(y_test, y_hat))
    errors.append(mse(y_test, y_hat))
```

```
print('Folds Accuracy:')
for i in acc_score:
    print('%.3f' %i)
print('Average Accuracy: %.3f' % (np.mean(acc_score)))

print('\nFolds MSE:')
for i in errors:
    print('%.3f' %i)
print('Average MSE: %.3f' % (np.mean(errors)))

# 10fold
kf = KFold(n_splits=10, random_state=None)

acc_score = []
errors = []

for train_index , test_index in kf.split(x):

    w, b = LinearRegr_grad(x_train, y_train, epochs=2000)
    y_hat = w * x_test + b

    acc_score.append(accur(y_test, y_hat))
    errors.append(mse(y_test, y_hat))

#%%10 fold

kf = KFold(n_splits=10, random_state=None)

acc_score = []
errors = []

for train_index , test_index in kf.split(x):



    w, b = LinearRegr_grad(x_train, y_train, epochs=2000)
    y_hat = w * x_test + b

    acc_score.append(accur(y_test, y_hat))
    errors.append(mse(y_test, y_hat))
```

```
print('Folds Accuracy:')
for i in acc_score:
    print('%.3f' %i)
print('Average Accuracy: %.3f' % (np.mean(acc_score)))

print('\nFolds MSE:')
for i in errors:
    print('%.3f' %i)
print('Average MSE: %.3f' % (np.mean(errors)))

# 10fold
kf = KFold(n_splits=10, random_state=None)

acc_score = []
errors = []

for train_index , test_index in kf.split(x):

    w, b = LinearRegr_grad(x_train, y_train, epochs=2000)
    y_hat = w * x_test + b

    acc_score.append(accur(y_test, y_hat))
    errors.append(mse(y_test, y_hat))

#%%models with package

df1 = shuffle(df1)
y = df1['livingSpace'].values
x = df1['noRooms'].values


cv = KFold(n_splits=5, random_state=1, shuffle=True)

model = LinearRegression()

acc = cross_val_score(model, x, y, cv=cv, n_jobs=-1)
error = -
cross_val_score(model, x, y, cv=cv, scoring='neg_mean_squared_error', n_jo
bs=-1)

print('Folds Accuracy:')
for i in acc:
    print('%.3f' %i)
print('Average Accuracy: %.3f' % (np.mean(acc)))
```

```
print('\nFolds MSE:')
for i in error:
    print('%.3f' %i)
print('Average MSE: %.3f' % (np.mean(error)))


cv = KFold(n_splits=5, random_state=1, shuffle=True)

rdg = Ridge()

acc = cross_val_score(rdg, x, y, cv=cv, n_jobs=-1)
error = cross_val_score(rdg, x, y, cv=cv, scoring='neg_mean_squared_error'
, n_jobs=-1)

print('Folds Accuracy:')
for i in acc:
    print('%.3f' %i)
print('Average Accuracy: %.3f' % (np.mean(acc)))

print('\nFolds MSE:')
for i in error:
    print('%.3f' %i)
print('Average MSE: %.3f' % (np.mean(error)))
cv = KFold(n_splits=5, random_state=1, shuffle=True)

lso = Lasso()

acc = cross_val_score(lso, x, y, cv=cv, n_jobs=-1)
error = -
cross_val_score(lso, x, y, cv=cv, scoring='neg_mean_squared_error', n_jobs
=-1)

print('Folds Accuracy:')
for i in acc:
    print('%.3f' %i)
print('Average Accuracy: %.3f' % (np.mean(acc)))

print('\nFolds MSE:')
for i in error:
    print('%.3f' %i)
print('Average MSE: %.3f' % (np.mean(error)))
```

Code for part 3:

```
#%%importing data and packages

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import precision_recall_fscore_support
from sklearn.utils import shuffle
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
from sklearn.decomposition import PCA
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate

from numpy import mean
from numpy import std


df2 = pd.read_csv (r'C:\Users\Asus\Desktop\Data Mining\02_assignment\data\
part3\train.csv')
#%%Understanding data for the second dataset

#to get a better understanding of how our data looks
df2.shape
df2.info() #no null data here
df2.describe()
df2.head(10)
df2.columns
#duplicated datapoints
df2.duplicated(keep=False).any() #no duplicates found

#%%normalization of numeric data??? do we need it
```

```
for cols in df2.columns:
    if df2[cols].dtype == 'int64' or df2[cols].dtype == 'float64':
         if cols != 'price_range':
              df2[cols] = ((df2[cols] - df2[cols].mean())/(df2[cols].std()
))


#%%spliting features and target
df2 = shuffle(df2)
x = df2.drop('price_range', axis=1)
y = df2['price_range']

#splitting test & train 80-20

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, r
andom_state=101)

#logistic regression for mobile cost classification 4 class
model = LogisticRegression()
model.fit(x_train,y_train)
predict = model.predict(x_test)

print(classification_report(y_test, predict))

#how are the 4 classes of price distributed in test and train
y_test.value_counts()
y_train.value_counts()
y_train

#2-class start
#%% working with two classes 0 and 1 for price range
#4 classes into 2 by adding all >0 into class 1
def edit_priceclass(x):
    if x != 0:
        return 1
    else:
        return x

y_test = y_test.apply(edit_priceclass)
y_train = y_train.apply(edit_priceclass)

y_test.value_counts()
y_train.value_counts()

#logistic regression for mobile cost classification 2 class
```

```
model = LogisticRegression()
model.fit(x_train,y_train)
predict = model.predict(x_test)

print(classification_report(y_test, predict))

#%%balancing the data and running regression with balances classes for 2-
class
y.value_counts()

ny = list()
for i in y:
    if i==0:
        ny.append(0)
    else:
        ny.append(1)
ny = np.array(ny)

for i in range(2):
    print(i, '-> ', np.count_nonzero(ny == i))

oversample = SMOTE(sampling_strategy='minority')
sm_x ,sm_y = oversample.fit_resample(x,ny)

for i in range(2):
    print(i, '-> ', np.count_nonzero(sm_y == i))

x_train, x_test, y_train, y_test = train_test_split(sm_x, sm_y, test_size=
0.2, random_state=101)

model = LogisticRegression()
model.fit(x_train,y_train)
predictions = model.predict(x_test)
print(classification_report(y_test, predictions))

#2class end


#%% Forward selection (https://pythonhealthcare.org/2020/01/04/feature-
selection-2-model-forward-selection/)

#standardising data
def standardise_data(x_train, x_test):

    # Initialise a new scaling object for normalising input data
```

```
    sc = StandardScaler()
    # Set up the scaler just on the training set
    sc.fit(x_train)
    # Apply the scaler to the training and test sets
    train_std = sc.transform(x_train)
    test_std = sc.transform(x_test)

    return train_std, test_std

# Create list to store accuracies and chosen features
roc_auc_by_feature_number = []
chosen_features = []

# Initialise chosen features list and run tracker
available_features = list(x)
run = 0
number_of_features = len(list(x))

# Loop through feature list to select next feature
while len(available_features)> 0:

    # Track and pront progress
    run += 1
    print ('Feature run {} of {}'.format(run, number_of_features))

    # Convert DataFrames to NumPy arrays
    y_np = y.values

    # Reset best feature and accuracy
    best_result = 0
    best_feature = ''

    # Loop through available features
    for feature in available_features:

        # Create copy of already chosen features to avoid orginal being ch
anged
        features_to_use = chosen_features.copy()
        # Create a list of features from features already chosen + 1 new f
eature
        features_to_use.append(feature)
        # Get data for features, and convert to NumPy array
        X_np = x[features_to_use].values

        # Set up lists to hold results for each selected features
```

```
        test_auc_results = []

        # Set up k-fold training/test splits
        number_of_splits = 5
        skf = StratifiedKFold(n_splits = number_of_splits)
        skf.get_n_splits(X_np, y)

        # Loop through the k-fold splits
        for train_index, test_index in skf.split(X_np, y_np):

            # Get X and Y train/test
            X_train, X_test = X_np[train_index], X_np[test_index]
            y_train, y_test = y[train_index], y[test_index]

            # Get X and Y train/test
            X_train_std, X_test_std = standardise_data(X_train, X_test)

            # Set up and fit model
            model = LogisticRegression(solver='lbfgs')
            model.fit(X_train_std,y_train)

            # Predict test set labels
            y_pred_test = model.predict(X_test_std)

            # Calculate accuracy of test sets
            accuracy_test = np.mean(y_pred_test == y_test)

            # Get ROC AUC
            probabilities = model.predict_proba(X_test_std)
            probabilities = probabilities[:, 1] # Probability of 'survived
'
            fpr, tpr, thresholds = roc_curve(y_test, probabilities, pos_la
bel=3)

            roc_auc = auc(fpr, tpr)
            test_auc_results.append(roc_auc)

        # Get average result from all k-fold splits
        feature_auc = np.mean(test_auc_results)

        # Update chosen feature and result if this feature is a new best
        if feature_auc > best_result:
            best_result = feature_auc
            best_feature = feature
```

```
    # Add mean accuracy and AUC to record of accuracy by feature number
    roc_auc_by_feature_number.append(best_result)
    chosen_features.append(best_feature)
    available_features.remove(best_feature)

# Put results in DataFrame
results = pd.DataFrame()
results['feature to add'] = chosen_features
results['ROC AUC'] = roc_auc_by_feature_number

results

#determining which features to use for log regr
results['feature to add'][results['ROC AUC']>0.540].values

#features we will use for model
x_fsel = x[['mobile_wt', 'fc', 'pc', 'battery_power', 'px_width','touch_sc
reen']]

x_train, x_test, y_train, y_test = train_test_split(x_fsel, y, test_size=0
.2, random_state=101)

#logistic regr on picked features by forward selection
model = LogisticRegression()
model.fit(x_train,y_train)
predictions = model.predict(x_test)
print(classification_report(y_test, predictions))

#%% log regr with PCA

pca = PCA(n_components=6)
pca.fit(x)
x_pca = pca.transform(x)

#spliting test and train with pca in mind
x_train, x_test, y_train, y_test = train_test_split(x_pca, y, test_size=0.
2, random_state=101)

#log regr for pca picked features
model = LogisticRegression()
model.fit(x_train,y_train)
predictions = model.predict(x_test)
print(classification_report(y_test, predictions))

#%%backward selection for features
```

```
# Create list to store accuracies and chosen features
roc_auc_by_feature_number = []
chosen_features = []

# Initialise chosen features list and run tracker
available_features = list(x)
run = 0
number_of_features = len(list(x))

# Create initial reference performance
reference_auc = 1.0 # used to compare reduction in AUC

# Loop through feature list to select next feature
while len(available_features)> 1:

    # Track and pront progress
    run += 1
    print ('Feature run {} of {}'.format(run, number_of_features-1))

    # Convert DataFrames to NumPy arrays
    y_np = y.values

    # Reset best feature and accuracy
    best_result = 1.0
    best_feature = ''

    # Loop through available features
    for feature in available_features:

        # Create copy of already chosen features to avoid orginal being ch
anged
        features_to_use = available_features.copy()
        # Create a list of features to use by removing 1 feature
        features_to_use.remove(feature)
        # Get data for features, and convert to NumPy array
        X_np = x[features_to_use].values

        # Set up lists to hold results for each selected features
        test_auc_results = []

        # Set up k-fold training/test splits
        number_of_splits = 5
        skf = StratifiedKFold(n_splits = number_of_splits)
        skf.get_n_splits(X_np, y)
```

```
        # Loop through the k-fold splits
        for train_index, test_index in skf.split(X_np, y_np):

            # Get X and Y train/test
            X_train, X_test = X_np[train_index], X_np[test_index]
            y_train, y_test = y[train_index], y[test_index]

            # Get X and Y train/test
            X_train_std, X_test_std = standardise_data(X_train, X_test)

            # Set up and fit model
            model = LogisticRegression(solver='lbfgs')
            model.fit(X_train_std,y_train)

            # Predict test set labels
            y_pred_test = model.predict(X_test_std)

            # Calculate accuracy of test sets
            accuracy_test = np.mean(y_pred_test == y_test)

            # Get ROC AUC
            probabilities = model.predict_proba(X_test_std)
            probabilities = probabilities[:, 1] # Probability of 'survived
'
            fpr, tpr, thresholds = roc_curve(y_test, probabilities, pos_la
bel=3)
            roc_auc = auc(fpr, tpr)
            test_auc_results.append(roc_auc)

        # Get average result from all k-fold splits
        feature_auc = np.mean(test_auc_results)

        # Update chosen feature and result if this feature is a new best
        # We are looking for the smallest drop in performance
        drop_in_performance = reference_auc - feature_auc
        if drop_in_performance < best_result:
            best_result = drop_in_performance
            best_feature = feature
            best_auc = feature_auc

    # k-fold splits are complete
    # Add mean accuracy and AUC to record of accuracy by feature number
    roc_auc_by_feature_number.append(best_auc)
    chosen_features.append(best_feature)
```

```
    available_features.remove(best_feature)
    reference_auc = best_auc

# Add last remaining feature
chosen_features += available_features
roc_auc_by_feature_number.append(0)

# Put results in DataFrame
# Reverse order of lists with [::-1] so best features first
results = pd.DataFrame()
results['feature removed'] = chosen_features[::-1]
results['ROC AUC'] = roc_auc_by_feature_number[::-1]
results

#determining which features to use for log regr
results['feature removed'][results['ROC AUC']>0.54].values

x_bsel = x[['four_g', 'blue', 'int_memory', 'wifi']]

x_train, x_test, y_train, y_test = train_test_split(x_bsel, y, test_size=0
.2, random_state=101)

#log regr for backward selection picked features
model = LogisticRegression()
model.fit(x_train,y_train)
predictions = model.predict(x_test)
print(classification_report(y_test, predictions))

#%%Kfold cross validation for the second dataset
from sklearn.model_selection import cross_validate
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score
, roc_auc_score, precision_score

# prepare the cross-validation procedure
k=5
cv = KFold(n_splits=k, random_state=None)


# create model
clf = LogisticRegression()

scoring = {'accuracy': 'accuracy',
           'recall': 'recall',
           'precision': 'precision',
           'roc_auc': 'roc_auc'}
```

```
# evaluate model
cross_val_scores = cross_validate(clf, x_train, y_train, cv=cv, scoring=sc
oring)

# report performance
print('Accuracy: %.3f (%.3f)' % (mean(cross_val_scores), std(cross_val_sco
res)))
```