Practical 10 Solutions

Jumping Rivers

Question 1 - Titanic

We're going to try and better the model prediction survival in the notes (shouldn't be hard!). The following code will load the data in and take a look at it

```
import pandas as pd
import jrpyml
titanic = jrpyml.datasets.load_titanic()
titanic.head()
```

```
##    PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
## 0            1         0       3  ...   7.2500   NaN         S
## 1            2         1       1  ...  71.2833   C85         C
## 2            3         1       3  ...   7.9250   NaN         S
## 3            4         1       1  ...  53.1000  C123         S
## 4            5         0       3  ...   8.0500   NaN         S
##
## [5 rows x 12 columns]
```

a) Set up your X_train and y_train objects such that your response variable is `Survived` and the one predictor variable is `Pclass`.

```
y_train = titanic["Survived"]
X_train = titanic[["Pclass"]]
```

b) `Pclass` represents the class of the persons room on the titanic. Should this be a categoric or a numeric variable? What data pre-processing should you therefore be using?

```
# Categoric so OneHotEncoding
from sklearn.preprocessing import OneHotEncoder
```

c) Write a pipeline the preprocesses the data in the correct way, then fits a regression model and then fit the model to your data.

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline

model = Pipeline([
 ('pre', OneHotEncoder()),
 ('logis', LogisticRegression(class_weight = 'balanced'))
])
model.fit(X_train, y_train)
```

d) For each class, what is the predicted category of survival and the corresponding probability for that category?

```
new_values = pd.DataFrame({
    "Pclass":[1,2,3]
})
model.predict(new_values)
```

```
## array([1, 1, 0])
```

```
model.predict_proba(new_values)
```

```
## array([[0.26831696, 0.73168304],
##        [0.42673767, 0.57326233],
##        [0.68209721, 0.31790279]])
```

e) Overall, how many predictions did we get correct?

```
from sklearn.metrics import accuracy_score
y_pred = model.predict(X_train)
accuracy_score(y_train,y_pred)
```

```
## 0.665266106442577
```

f) Of those that survived, what proportion were actually classified that way?

```
from sklearn.metrics import  recall_score
recall_score(y_train,y_pred,pos_label=1) # tp/(tp + fp)
```

```
## 0.7068965517241379
```

g) The following code will perform 10-fold cross validation on the data and return the accuracy. Make it return the precision and recall

```
from sklearn.model_selection import cross_validate
from sklearn.metrics import make_scorer
import pandas as pd

acc = make_scorer(accuracy_score)

output = cross_validate(model,X_train,y_train,scoring={
  'acc' : acc
}, cv = 10, return_train_score=False)
```

```
from sklearn.model_selection import cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score
```

```
import pandas as pd

acc = make_scorer(accuracy_score)

def precision(y_true,y_pred):
  return precision_score(y_true,y_pred,pos_label = 1)

def recall(y_true,y_pred):
  return recall_score(y_true, y_pred, pos_label = 1)

prec = make_scorer(precision)
rec = make_scorer(recall)
output = cross_validate(model,X_train,y_train,scoring={
  'acc' : acc,
  'prec' : prec,
  'rec' : rec
}, cv = 10, return_train_score=False)
```

What is the average test accuracy, precision and recall? What does this tell you about the model?

Question 2 - Advancing titanic

To attempt to improve the model, we want to inclue Age in the model.

a) Set up your `X_train` model appropriately

```
X_train = titanic[["Age", "Pclass"]]
```

b) Using `ColumnTransformer()`, `StandardScaler()` and `OneHotEncoder()`, set up an appropriate preprocessing object, then include it in a model pipeline and fit the model to the data

```
from sklearn.compose import ColumnTransformer
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler, OneHotEncoder

numeric_features = ['Age']
categorical_features = ['Pclass']

preprocessor = ColumnTransformer(
  transformers=[
    ('num', StandardScaler(), numeric_features),
    ('cat', OneHotEncoder(),  categorical_features)
  ]
)
```

```
model = Pipeline(
  steps = [
    ('preprocess', preprocessor),
    ('regression', linear_model.LogisticRegression())
  ]
)

model.fit(X_train, y_train)
```

c) The following code will set up a DataFrame of peoples ages and pclasses. Use your model to predict whether these people would survive.

```
import numpy as np
Age = np.repeat([10,20,30,40,50,60], repeats = 3)
Pclass = np.array([1,2,3]*6)
new_values = pd.DataFrame({
    "Age":Age,
    "Pclass":Pclass
})

new_values["pred"] = model.predict(new_values)
```

```
## /home/theo/anaconda3/lib/python3.7/site-packages/sklearn/pipeline.py:605: DataConversionWarning: Data
##   res = transformer.transform(X)
```

d) We could plot the new persons like so.

```
import seaborn as sns
sns.scatterplot(x = "Age", y = "Pclass", hue = "pred", data = new_values)
```

What is this graph showing? What does this say about the relationship between Age, Pclass and Survived?

e) Just like in part g) of the previous question, the following code will perform 10-fold criss validation on the new model.

```
from sklearn.model_selection import cross_validate
from sklearn.metrics import make_scorer
import pandas as pd

acc = make_scorer(accuracy_score)

def precision(y_true,y_pred):
  return precision_score(y_true,y_pred,pos_label = 1)
```

```
def recall(y_true,y_pred):
  return recall_score(y_true, y_pred, pos_label = 1)

prec = make_scorer(precision)
rec = make_scorer(recall)
output = cross_validate(model,X_train,y_train,scoring={
  'acc' : acc,
  'prec' : prec,
  'rec' : rec
}, cv = 10, return_train_score=False)
```

How does the test accuracy compare to the previous model? Have we improved results?