Module 3 Notebook

July 20, 2021

1 Module 3 Homework Assignments

1.1 Filipp Krasovsky, 7-19-2021

[421]:
```
import json
import re
import seaborn as sb
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.preprocessing import OrdinalEncoder
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import plot_confusion_matrix, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import Perceptron
from sklearn.impute import SimpleImputer
from sklearn.model_selection import cross_val_score
```

[400]:
```
#iterate over the json records and append them to a list.
recordlist=[]
with open('modcloth_final_data.json') as f:
    for obj in f:
        thisRecord = json.loads(obj)
        recordlist.append(thisRecord)
```

[401]:
```
#convert into a dataframe
modcloth_data = pd.DataFrame(recordlist)
modcloth_data = modcloth_data.dropna(subset=['quality'])
```

[402]:
```python
#indicate our response variable
modcloth_data['labels'] = modcloth_data['quality']
modcloth_data           = modcloth_data.drop('quality',axis=1)
modcloth_data.head()
```

[402]:
```
  item_id waist  size cup size hips bra size category bust   height  \
0  123373    29     7        d   38       34      new   36  5ft 6in
1  123373    31    13        b   30       36      new  NaN  5ft 2in
2  123373    30     7        b  NaN       32      new  NaN  5ft 7in
3  123373   NaN    21     dd/e  NaN      NaN      new  NaN      NaN
4  123373   NaN    18        b  NaN       36      new  NaN  5ft 2in

         user_name        length    fit user_id shoe size shoe width  \
0            Emily    just right  small  991571       NaN        NaN
1  sydneybraden2001    just right  small  587883       NaN        NaN
2            Ugggh slightly long  small  395665      9.00        NaN
3      alexmeyer626    just right    fit  875643       NaN        NaN
4       dberrones1 slightly long  small  944840       NaN        NaN

  review_summary review_text  labels
0            NaN         NaN     5.0
1            NaN         NaN     3.0
2            NaN         NaN     2.0
3            NaN         NaN     5.0
4            NaN         NaN     5.0
```

[403]:
```python
#convert numeric variables
for variable in ['waist','size','hips','bra size','shoe size']:
    modcloth_data[variable] = pd.to_numeric(modcloth_data[variable])
```

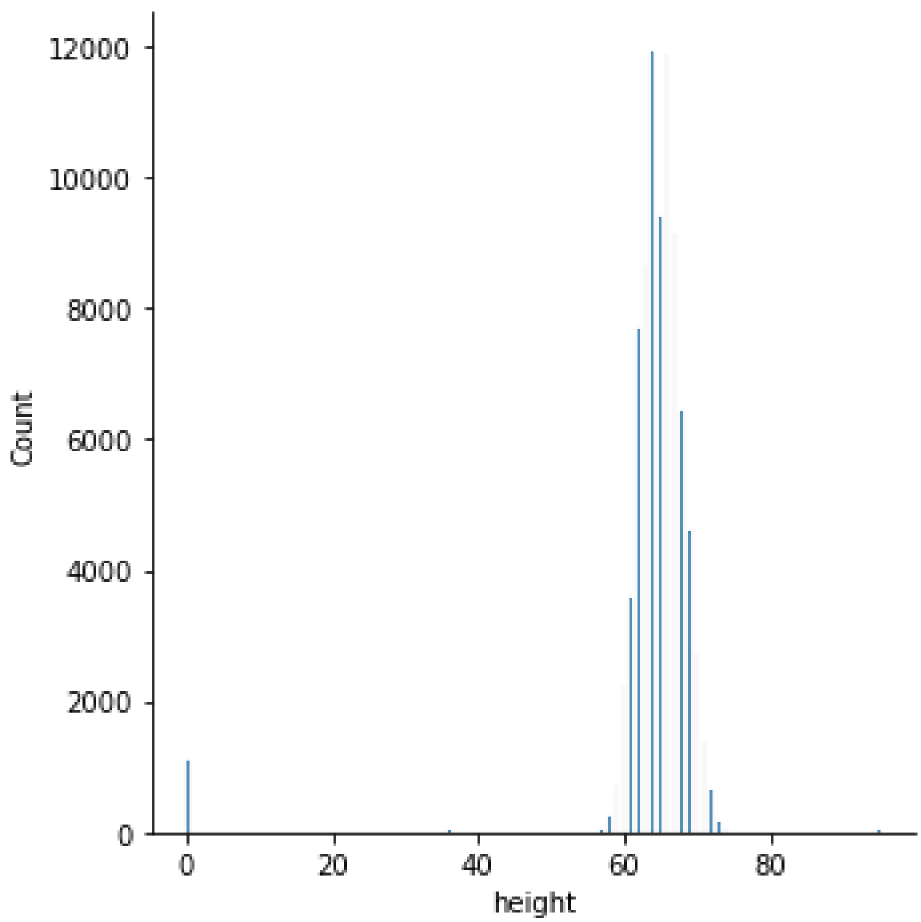
[404]:
```python
#convert height into a categorical variable using the pd.apply function.
#we have two possibilities here - a nan or a string.
#if nan => 0
#if !nan => split the string by the empty space between ft and in. to get a␣
 ↪vector (f,i)
#the final value will be f*12 + i

def toInches(height):
    if (np.nan_to_num(height)==0):
        return 0
    else:
        args = height.split()
        out  = 0
        out = out+(pd.to_numeric(re.sub('[^0-9]','', args[0]))*12)

        #make sure to check we have an inches component in our string before␣
 ↪casting.
```

```
        if (len(args)==2):
            out = out+pd.to_numeric(re.sub('[^0-9]','', args[1]))

        return (out)

modcloth_data['height'] = modcloth_data.apply(lambda row:␣
 ↪toInches(row['height']),axis=1)
```

[405]:
```
#we next apply this to the bust variable
modcloth_data['bust'] = modcloth_data.apply(lambda row:␣
 ↪toInches(row['bust']),axis=1)
```

[225]:
```
#plot height
sb.displot(pd.Series(modcloth_data['height']))
```

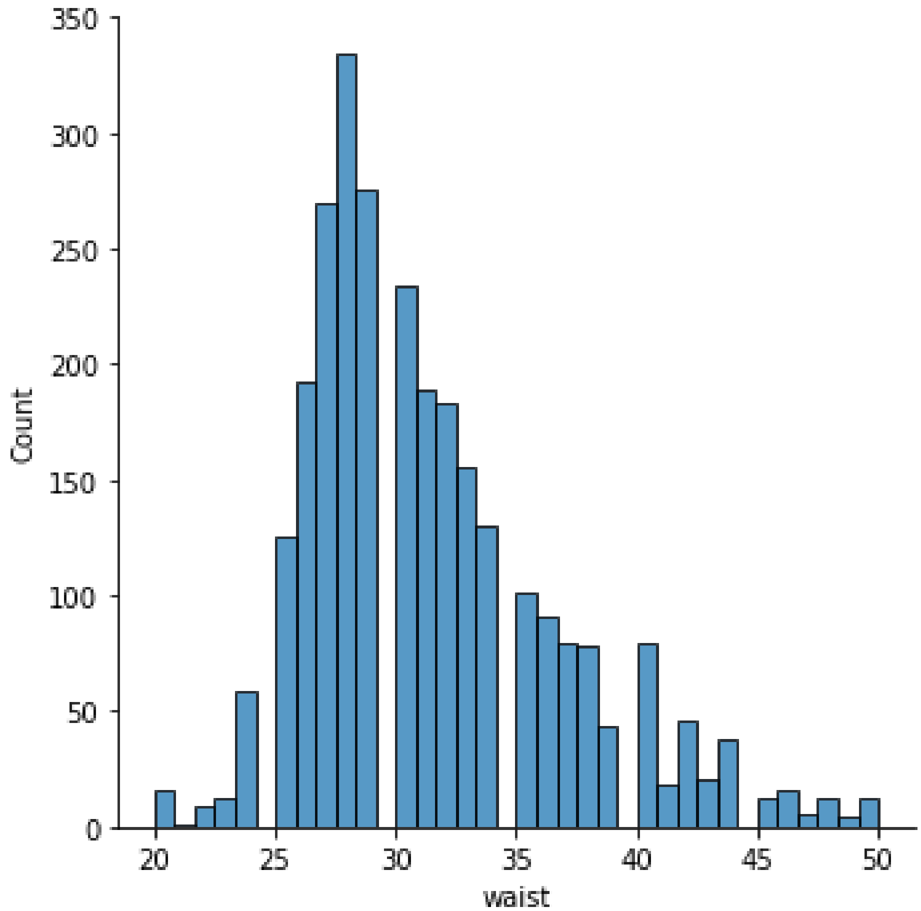
[225]: <seaborn.axisgrid.FacetGrid at 0x1c68b5752b0>



Observations seem to suggest that height is relatively normally distributed with some outliers at zero.

[226]:
```
sb.displot(modcloth_data['waist'])
```

[226]: <seaborn.axisgrid.FacetGrid at 0x1c68b575190>



2 Categorical Data for Reviews

[227]:
```
#define the labels as our Y variable
y = modcloth_data['labels']
```

[228]:
```
#turn cup size, length, and category into categorical OHE vars:
#use the oneHotEncoder to transform our variables and get label names

cat_feat = modcloth_data[['bra size','length','category']]

ohe = OneHotEncoder(sparse='False')
feature_array = ohe.fit_transform(cat_feat).toarray()
feature_labels= ohe.categories_
```

```
#convert labels into one array
feature_labels = np.concatenate(feature_labels)
#combine labels and data
cat_feat = pd.DataFrame(feature_array,columns = feature_labels)
```

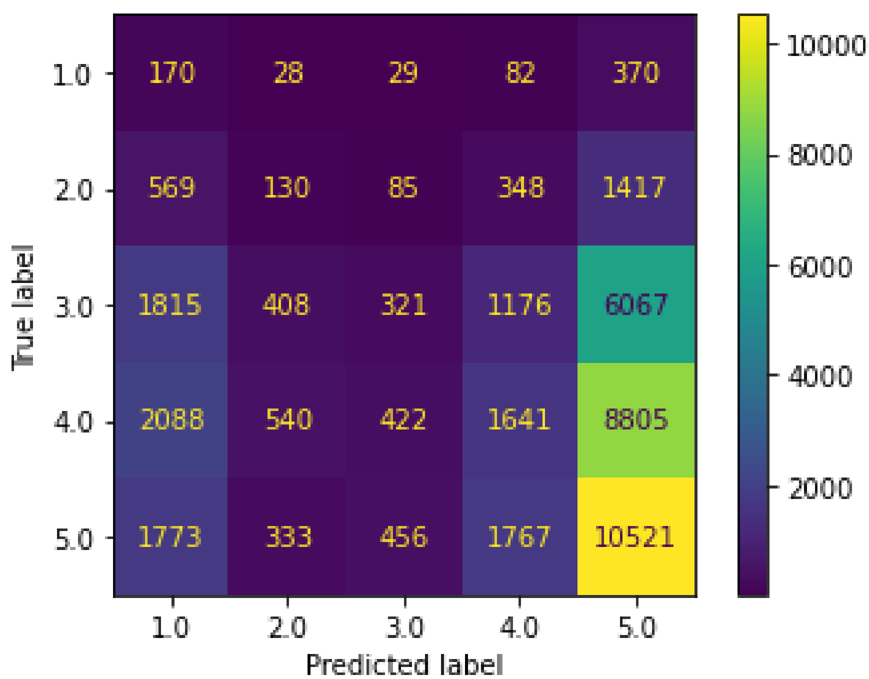
[229]:
```
balanced_model = make_pipeline(Perceptron(class_weight='balanced'))
unbalanced_model = make_pipeline(Perceptron())
```

[230]:
```
X_train, X_test, y_train, y_test = train_test_split(cat_feat, y, test_size=0.
↪50, random_state=42)
```

[231]:
```
balanced_model.fit(X_train,y_train)
unbalanced_model.fit(X_train,y_train)

#predict
balanced_pred = balanced_model.predict(X_test)
unbalanced_pred=unbalanced_model.predict(X_test)
```

[232]:
```
#Plot for balanced perceptron
plot_confusion_matrix(balanced_model,X_train,y_train)
```

[232]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c68ad11d90>

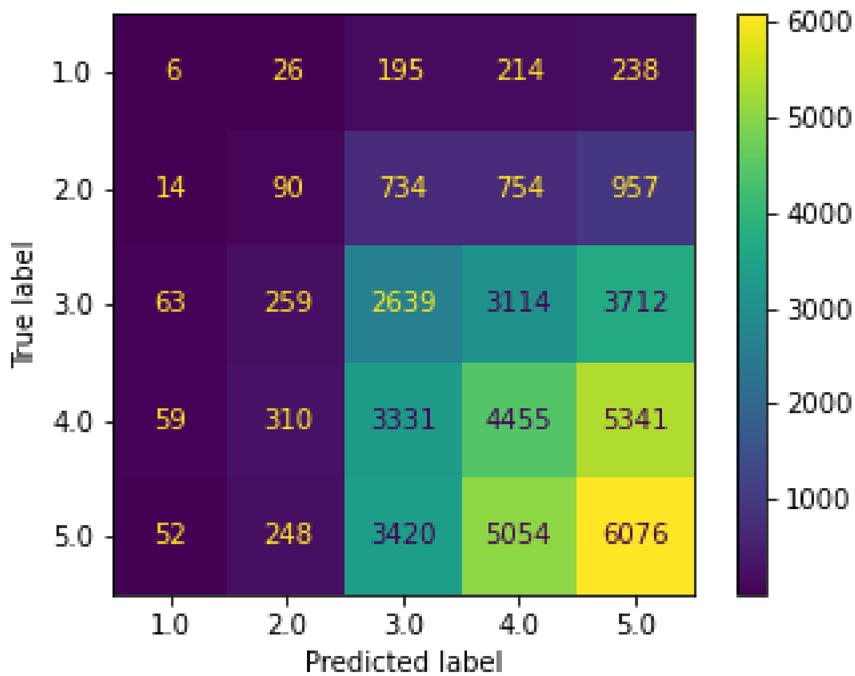


[233]:
```
print(classification_report(balanced_pred,y_test))
```

```
              precision    recall  f1-score   support

         1.0       0.23      0.02      0.04      6266
         2.0       0.05      0.10      0.07      1355
         3.0       0.03      0.23      0.06      1351
         4.0       0.12      0.32      0.18      5197
         5.0       0.71      0.38      0.50     27192

    accuracy                           0.31     41361
   macro avg       0.23      0.21      0.17     41361
weighted avg       0.52      0.31      0.36     41361
```

[234]:
```
#Plot for unbalanced perceptron
plot_confusion_matrix(unbalanced_model,X_train,y_train)
```

[234]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c688fe8610>



[235]:
```
print(classification_report(unbalanced_pred,y_test))
```

```
              precision    recall  f1-score   support

         1.0       0.01      0.03      0.01       213
         2.0       0.03      0.08      0.04       888
         3.0       0.27      0.25      0.26     10359
```

```
         4.0       0.33      0.32      0.33     13762
         5.0       0.40      0.37      0.38     16139

    accuracy                           0.32     41361
   macro avg       0.21      0.21      0.21     41361
weighted avg       0.33      0.32      0.33     41361
```

3 Categorical & Numeric Features

[406]:
```
#here, we combine numerical features with the cat_feat set.
#before we do so, we're going to use a simple imputer.
#then, we try the normalizing scaler and the standardizing scaler.
#then we put them through our perceptron.

#we can eliminate item_id, user_name, user_id, shoe width, review summary, and␣
 ↪review text.
num_feat = pd.DataFrame(modcloth_data[['waist','size','hips','bra␣
 ↪size','bust','height','shoe size']])
cat_feat = modcloth_data[['cup size','length','category']]

#re-encode.
ohe = OneHotEncoder(sparse='False')
feature_array = ohe.fit_transform(cat_feat).toarray()
feature_labels= ohe.categories_
#convert labels into one array
feature_labels = np.concatenate(feature_labels)
#combine labels and data
cat_feat = pd.DataFrame(feature_array,columns = feature_labels)

y        = modcloth_data['labels']
```

[417]:
```
#combine
cat_feat.reset_index()
num_feat.reset_index()
cat_feat = cat_feat.loc[~cat_feat.index.duplicated(keep='first')]
num_feat = num_feat.loc[~num_feat.index.duplicated(keep='first')]
X_cols = pd.concat([cat_feat,num_feat],axis=1).columns
X = pd.DataFrame(np.hstack([cat_feat,num_feat]))
X.columns = X_cols
```

[424]:
```
#create a pipeline with imputation
imputer_obj = SimpleImputer(missing_values=np.NaN, strategy='mean')
impute_pipeline = make_pipeline(imputer_obj,Perceptron(class_weight='balanced'))
impute_pipeline.fit(X,y)
```

```
impute_accuracy = cross_val_score(impute_pipeline, X, y,␣
↪cv=5,scoring='accuracy')
```

[428]:
```
imputed_X = pd.DataFrame(imputer_obj.fit_transform(X))
imputed_X.columns = X_cols
```

[437]:
```
#pipelines with normalization and standard scaling
normalizer = preprocessing.Normalizer()
standard_scaler = preprocessing.StandardScaler()

norm_pipeline = make_pipeline(normalizer,Perceptron(class_weight='balanced'))
norm_pipeline.fit(imputed_X,y)
scale_pipeline=␣
↪make_pipeline(standard_scaler,Perceptron(class_weight='balanced'))
scale_pipeline.fit(imputed_X,y)
```

[437]:
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('perceptron', Perceptron(class_weight='balanced'))])
```

[441]:
```
#test accuracies
norm_accuracy = cross_val_score(norm_pipeline, imputed_X, y,␣
↪cv=5,scoring='accuracy')
scale_accuracy= cross_val_score(scale_pipeline,imputed_X, y,␣
↪cv=5,scoring='accuracy')
```

[453]:
```
def getSummary(name,scores):
    minScore = min(scores)
    maxScore = max(scores)
    meanScore= np.mean(scores)

    return([name,minScore,maxScore,meanScore])
```

[461]:
```
results = pd.DataFrame([
    getSummary("Impute Pipeline",impute_accuracy),
    getSummary("Norm. Pipeline",norm_accuracy),
    getSummary("Scale Pipeline",scale_accuracy)
])

results.columns = ['name','min','max','avg']
```

[462]:
```
results
```

[462]:

	name	min	max	avg
0	Impute Pipeline	0.073622	0.328941	0.176494
1	Norm. Pipeline	0.073561	0.358075	0.252131
2	Scale Pipeline	0.212766	0.260397	0.236164

4 Text Data Analysis

[496]:
```
corpus = modcloth_data[['review_text','labels']]
#remove all outliers - data with an NA
corpus = corpus.dropna()
```

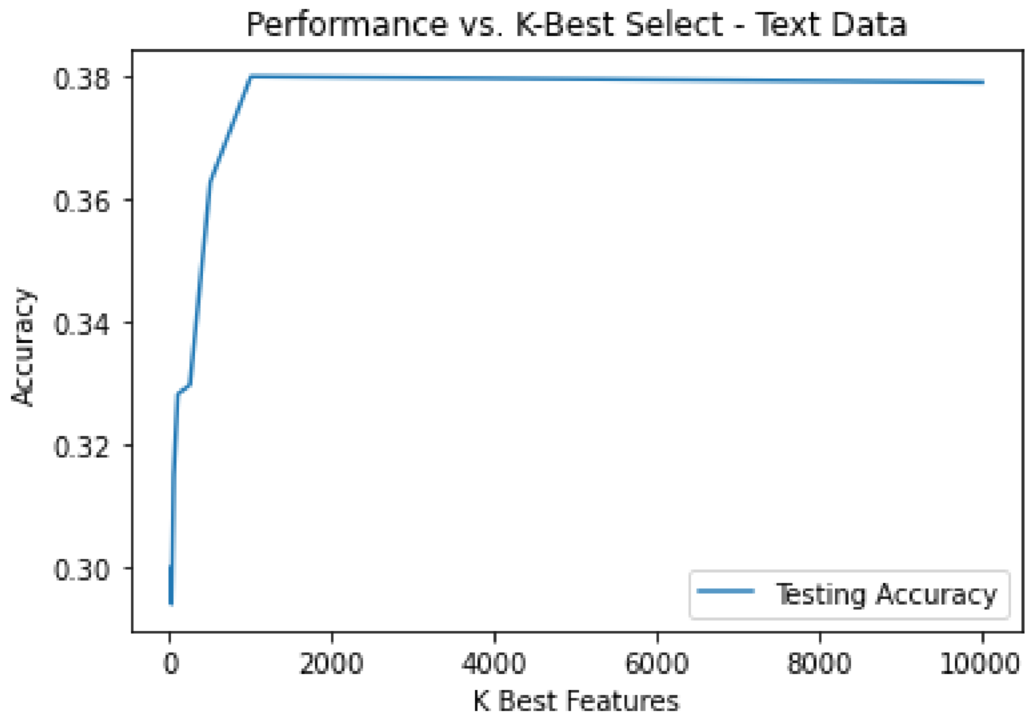
[497]:
```
#Pass this new variable to sklearn'sTfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

[531]:
```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus['review_text'])
X = pd.DataFrame.sparse.from_spmatrix(X)
X.columns = vectorizer.get_feature_names()
```

[501]:
```
X_train, X_test, y_train, y_test = train_test_split(X, corpus['labels'],␣
↪test_size=0.20, random_state=42)
```

[577]:
```
pipeline = Perceptron()
accuracy = []
kbest = [10,25,50,100,250,500,1000,10000]
features = []
feature_names = X.columns

for i  in kbest:
    X_new = SelectKBest(chi2, k=i).fit(X_train, y_train)
    mask = X_new.get_support()
    new_features = []

    for bool, feature in zip(mask, feature_names):
        if bool:
            new_features.append(feature)

    features.append(new_features)

    X_newtest = X_new.transform(X_test)
    X_newtrain= X_new.transform(X_train)
    this_fit = pipeline.fit(X_newtrain,y_train)
    this_pred=this_fit.predict(X_newtest)
    accuracy.append(accuracy_score(this_pred,y_test))
```

[574]:
```
#plot performance for text data
plt.plot(kbest, accuracy, label = "Testing Accuracy")
plt.legend()
plt.xlabel("K Best Features")
plt.ylabel("Accuracy")
plt.title("Performance vs. K-Best Select - Text Data")
plt.show()
```



[580]:
```
#top ten words
features[0]
```

[580]:
```
['cheap',
 'disappointed',
 'love',
 'perfect',
 'poor',
 'returned',
 'ripped',
 'terrible',
 'thin',
 'was']
```