worksheet_14

March 24, 2024

1 Worksheet 14

Name: Daniyal Ahmed
UID: U11469883

1.0.1 Topics

- Naive Bayes
- Model Evaluation

1.0.2 Naive Bayes

Attribute A	Attribute B	Attribute C	Class
Yes	Single	High	No
No	Married	Mid	No
No	Single	Low	No
Yes	Married	High	No
No	Divorced	Mid	Yes
No	Married	Low	No
Yes	Divorced	High	No
No	Single	Mid	Yes
No	Married	Low	No
No	Single	Mid	Yes

a) Compute the following probabilities:

- P(Attribute A = Yes | Class = No)
- P(Attribute B = Divorced | Class = Yes)
- P(Attribute C = High | Class = No)
- P(Attribute C = Mid | Class = Yes)

A). $\frac{3}{7}$ B). $\frac{1}{3}$ C). $\frac{2}{7}$ D). 1

b) Classify the following unseen records:

- (Yes, Married, Mid)
- (No, Divorced, High)
- (No, Single, High)
- (No, Divorced, Low)

A). Class = No B). Class = No C). Class = No D.) Class = No

1.0.3 Model Evaluation

a) Write a function to generate the confusion matrix for a list of actual classes and a list of predicted classes

```
[2]: actual_class = ["Yes", "No", "No", "Yes", "No", "No", "Yes", "No", "No", "No"]
     predicted_class = ["Yes", "No", "Yes", "No", "No", "No", "Yes", "Yes", "Yes",␣
      ↪"No"]

     import numpy as np

     def confusion_matrix(actual, predicted):
         TP = 0
         FN = 0
         FP =0
         TN = 0
         for i in range(len(actual)):
             if(actual[i] == "Yes" and predicted[i] == "Yes"):
                 TP += 1

             if(actual[i] == "Yes" and predicted[i] == "No"):
                 FN += 1

             if(actual[i] == "No" and predicted[i]== "No"):
                 FP +=1

             if(actual[i] == "No" and predicted[i] == "Yes"):
                 TN +=1

         return np.array([[TP,FN],[FP,TN]])

     print(confusion_matrix(actual_class, predicted_class))
```

```
[[2 1]
 [4 3]]
```

b) Assume you have the following Cost Matrix:

	predicted = Y	predicted = N
actual = Y	-1	5
actual = N	10	0

What is the cost of the above classification?

43

c) Write a function that takes in the actual values, the predictions, and a cost matrix and outputs a cost. Test it on the above example.

```python
[3]: def cost(confusion_matrix, cost_matrix):
         cost = 0
         for i in range(len(confusion_matrix)):
             for j in range(len(confusion_matrix)):
                 cost += confusion_matrix[i][j] *cost_matrix[i][j]

         return cost
```

d) Implement functions for the following:

- accuracy
- precision
- recall
- f-measure

and apply them to the above example.

```python
[4]: def accuracy(confusion_matrix):
         total = 0
         for i in range(len(confusion_matrix)):
             total+= sum(confusion_matrix[i])

         return (confusion_matrix[0][0] + confusion_matrix[1][1])/total



     def precision(confusion_matrix):
         return confusion_matrix[0][0]/(confusion_matrix[0][0] +␣
      ↪confusion_matrix[1][0])

     def recall(confusion_matrix):
         return confusion_matrix[0][0] / (confusion_matrix[0][0] +␣
      ↪confusion_matrix[0][1])

     def f_measure(confusion_matrix):
         return 2 * (precision(confusion_matrix) * recall(confusion_matrix)) /␣
      ↪(precision(confusion_matrix) + recall(confusion_matrix))
```

1.1 Challenge (Midterm prep part 2)

In this exercise you will update your submission to the titanic competition.

a) First let's add new numerical features / columns to the datasets that might be related to the survival of individuals.

- `has_cabin` should have a value of 0 if the `cabin` feature is `nan` and 1 otherwise
- `family_members` should have the total number of family members (by combining `SibSp` and `Parch`)
- `title_type`: from the title extracted from the name, we will categorize it into 2 types: `common`

for titles that many passengers have, `rare` for titles that few passengers have. Map `common` to 1 and `rare` to 0. Describe what threshold you used to define `common` and `rare` titles and how you found it.

- `fare_type`: using Kmeans clustering on the fare column, find an appropriate number of clusters / groups of similar fares. Using the clusters you created, `fare_price` should be an ordinal variable that represents the expensiveness of the fare. For example if you split fare into 3 clusters (0 - 15, 15 - 40, and 40+) then the `fare_price` value should be `0` for `fare` values 0 - 15, `1` for 15 - 40, and `2` for 40+.
- Create an addition two numerical features of your invention that you think could be relevant to the survival of individuals.

Note: The features must be numerical because the sklearn `DecisionTreeClassifier` can only take on numerical features.

[5]:
```python
import pandas as pd
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
from copy import deepcopy

dataset = pd.read_csv('train.csv')

dataset['has cabin'] = dataset['Cabin'].apply(lambda x: 0 if pd.isnull(x) else␣
 ↪1)

dataset['family members'] = dataset['SibSp'] + dataset['Parch']

titles = {}
total = 0


for i in dataset['Name']:
    title = i.split(',')[1].split('.')[0].strip()
    if title in titles:
        titles[title] += 1
    else:
        titles[title] = 1

    total += 1

'''Overwhelming majority of the titles are Mr, Mrs, and Miss. about 92 %'''
print((titles['Mr']+titles['Mrs']+titles['Miss'])/total)

threshold = 1- (titles['Mr']+titles['Mrs']+titles['Miss'])/total


'''If it is greater than the threshold then is common title otherwise it is␣
 ↪rare title'''
```

```python
dataset['title_type'] = dataset['Name'].apply(lambda x: 1 if titles[x.
 ↪split(',')[1].split('.')[0].strip()]/total > threshold else 0)

#Easier to do this since it is a one dimensional array
ranges = {}
current_range = 15

ranges[current_range] = 0


fares = dataset['Fare'].tolist()

fares.sort()
for price in fares:
    if price > current_range:
        current_range += 15
        ranges[current_range] = 0
    else:
        ranges[current_range] += 1

print(ranges)

# Perform KMeans clustering
kmeans = KMeans(n_clusters=4, random_state=0).fit(dataset[['Fare']])

dataset['fare_type'] = kmeans.labels_


Ages = dataset['Age'].fillna(dataset['Age'].mean()).tolist()
PassengerId = dataset['PassengerId'].tolist()

for i in range(len(Ages)):
    Ages[i] = [Ages[i], PassengerId[i]]


ranges = {}

Ages.sort(key=lambda x: x[0])


Age_range = 10
ranges[Age_range] = 0
for Age in Ages:
    if Age[0] > Age_range:
        Age_range += 10
        ranges[Age_range] = 0
    else:
        ranges[Age_range] += 1



print(ranges)

# Perform KMeans clustering
kmeans = KMeans(n_clusters=5, random_state=0).fit(dataset[['Fare']])

dataset['Sex'] = dataset['Sex'].apply(lambda x: 1 if x == 'Male' else 0)


def age(x):
    if(0<=x<=10):
        return 0

    if(11<=x<=15):
        return 1
    if(16<=x<=20):
        return 2
    return 3



dataset['Age_range'] = dataset['Age'].apply(age)



dataset['Alone'] = dataset['family members'].apply(lambda x: 1 if x > 0 else 0)






'''The Way I found Rare and common titles was by grouping all the titles via a␣
 ↪dictionary method, and then seeing what titles repeat the most, the␣
 ↪overwhelming majority of titles
were Mr, Mrs. Miss which consisted of 92 % of titles so if any title was less␣
 ↪than or equal to 8% is was marked with rare, The features I decided to add␣
 ↪where if the person has any family
members on board is not we can infer that they might have come alone, secondly␣
 ↪I added age ranges for a ages 0-10, 11-15, 16-20 and 20 and above. These␣
 ↪features I feel are relevant to the survival of a
passenger '''
```

```
0.9248035914702581
```

```
{15: 458, 30: 198, 45: 62, 60: 48, 75: 24, 90: 39, 105: 3, 120: 14, 135: 3, 150:
4, 165: 8, 180: 0, 195: 0, 210: 0, 225: 1, 240: 3, 255: 1, 270: 5, 285: 0, 300:
0, 315: 0}
{10: 64, 20: 114, 30: 406, 40: 154, 50: 85, 60: 41, 70: 16, 80: 4}
```

```
/home/daniyal-ahmed/.local/lib/python3.11/site-
packages/sklearn/cluster/_kmeans.py:1412: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/home/daniyal-ahmed/.local/lib/python3.11/site-
packages/sklearn/cluster/_kmeans.py:1412: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

[5]:
```
'The Way I found Rare and common titles was by grouping all the titles via a
dictionary method, and then seeing what titles repeat the most, the overwhelming
majority of titles \nwere Mr, Mrs. Miss which consisted of 92 % of titles so if
any title was less than or equal to 8% is was marked with rare, The features I
decided to add where if the person has any family \nmembers on board is not we
can infer that they might have come alone, secondly I added age ranges for a
ages 0-10, 11-15, 16-20 and 20 and above. These features I feel are relevant to
the survival of a \npassenger '
```

b) Using a method covered in class, tune the parameters of a decision tree model on the titanic dataset (containing all numerical features including the ones you added above). Evaluate this model locally and report it's performance.

Note: make sure you are not tuning your parameters on the same dataset you are using to evaluate the model. Also explain how you know you are not overfitting to the training set.

[6]:
```python
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score





'''We will use the following features to predict survival: Age_range,␣
↪fare_type, title_type, has cabin, family members, has child, Pclass, Fare␣
↪Aka all the
numerial features'''
X= dataset[['Age_range', 'fare_type', 'title_type', 'has cabin', 'family␣
↪members', 'Alone', 'Pclass', 'Sex']]
y= dataset['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪random_state=42)
```

```



tree = DecisionTreeClassifier(criterion='gini' ,random_state=1, max_depth=6)
tree.fit(X_train, y_train)


predictions_dt = tree.predict(X_test)

accuracy = accuracy_score(y_test, predictions_dt)
print(f"Model Accuracy: {accuracy}")
```

```
Model Accuracy: 0.7085201793721974
```

c) Try reducing the dimension of the dataset and create a Naive Bayes model. Evaluate this model.

```
[7]: from sklearn.naive_bayes import GaussianNB
from sklearn.decomposition import PCA

'''Got PCA from the example code given below the challenge problem'''
pca = PCA(n_components=5)

X= dataset[['Age_range', 'fare_type', 'title_type', 'has cabin', 'family␣
 ↪members', 'Alone','Pclass', 'Sex']]
y= dataset['Survived']

#X= pca.fit_transform(X)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=42)




# Initialize the model, Assuming a normal distribution Since it can't possibly␣
 ↪be discrete
model = GaussianNB()

# Train the model using the training sets
model.fit(X_train, y_train)

predictions_nb = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions_nb)
print(f"Model Accuracy: {accuracy}")
```

```
Model Accuracy: 0.7085201793721974
```

d) Create an ensemble classifier using a combination of KNN, Decision Trees, and Naive Bayes models. Evaluate this classifier.

[8]:
```
from scipy.stats import mode
from sklearn.neighbors import KNeighborsClassifier

'''Had to do some googling since this part wasn't covered in last class, but I␣
↪first googled what an essemble
classifier is and found out that it was a way of combining models to get a␣
↪better result. I found this function from
scipy called mode, which takes the prediction of all three models and returns␣
↪the most common prediction. I then used
Its almost like a voting feature, the models vote on which one they think is␣
↪more likely to be correct and the most common'''



knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
predictions_knn = knn.predict(X_test)





predictions_combined = np.array([predictions_knn, predictions_dt,␣
↪predictions_nb])
predictions2, _ = mode(predictions_combined, axis=0)
accuracy = accuracy_score(y_test, predictions2.ravel())
print(f"Model Accuracy: {accuracy}")
```

```
Model Accuracy: 0.7219730941704036
```

e) Update your kaggle submission using the best model you created (best model means the one that performed the best on your local evaluation)

I used the Decision Tree model since it performed best locally

[11]:
```
#USERNAME:daniyala123

import pandas as pd
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
from copy import deepcopy

dataset = pd.read_csv('test.csv')

dataset['has cabin'] = dataset['Cabin'].apply(lambda x: 0 if pd.isnull(x) else␣
↪1)
```

```
dataset['family members'] = dataset['SibSp'] + dataset['Parch']

titles = {}
total = 0


for i in dataset['Name']:
    title = i.split(',')[1].split('.')[0].strip()
    if title in titles:
        titles[title] += 1
    else:
        titles[title] = 1

    total += 1

'''Overwhelming majority of the titles are Mr, Mrs, and Miss. about 92 %'''
print((titles['Mr']+titles['Mrs']+titles['Miss'])/total)

threshold = 1- (titles['Mr']+titles['Mrs']+titles['Miss'])/total


'''If it is greater than the threshold then is common title otherwise it is␣
 ↪rare title'''
dataset['title_type'] = dataset['Name'].apply(lambda x: 0 if titles[x.
 ↪split(',')[1].split('.')[0].strip()]/total > threshold else 1)

#Easier to do this since it is a one dimensional array
ranges = {}
current_range = 15

ranges[current_range] = 0


fares = dataset['Fare'].fillna(dataset['Fare'].mean()).tolist()
dataset['Fare'] = dataset['Fare'].fillna(dataset['Fare'].mean())
fares.sort()
for price in fares:
    if price > current_range:
        current_range += 15
        ranges[current_range] = 0
    else:
        ranges[current_range] += 1

print(ranges)

# Perform KMeans clustering
kmeans = KMeans(n_clusters=4, random_state=0).fit(dataset[['Fare']])
```

```
dataset['fare_type'] = kmeans.labels_


Ages = dataset['Age'].fillna(dataset['Age'].mean()).tolist()
PassengerId = dataset['PassengerId'].tolist()

for i in range(len(Ages)):
    Ages[i] = [Ages[i], PassengerId[i]]


ranges = {}

Ages.sort(key=lambda x: x[0])


Age_range = 10
ranges[Age_range] = 0
for Age in Ages:
    if Age[0] > Age_range:
        Age_range += 10
        ranges[Age_range] = 0
    else:
        ranges[Age_range] += 1



print(ranges)

# Perform KMeans clustering
kmeans = KMeans(n_clusters=5, random_state=0).fit(dataset[['Fare']])

dataset['Sex'] = dataset['Sex'].apply(lambda x: 1 if x == 'Male' else 0)



def age(x):
    if(0<=x<=10):
        return 0

    if(11<=x<=15):
        return 1
    if(16<=x<=20):
        return 2
    return 3
```

```python
dataset['Age_range'] = dataset['Age'].apply(age)




dataset['Alone'] = dataset['family members'].apply(lambda x: 1 if x > 0 else 0)


test = dataset[['Age_range', 'fare_type', 'title_type', 'has cabin', 'family␣
↪members', 'Alone','Pclass', 'Sex', ]]

predictions_knn = knn.predict(test)
predictions_nb = model.predict(test)
predictions_dt = tree.predict(test)
predictions_combined= [predictions_dt,predictions_nb,predictions_knn]

print(predictions_dt)
#Saving the CSV file
csv = pd.DataFrame({
    'PassengerId': dataset['PassengerId'],
    'Survived': predictions_dt
})

csv.to_csv('sub.csv', index=False)
```

```
0.9330143540669856
{15: 215, 30: 92, 45: 24, 60: 22, 75: 13, 90: 14, 105: 1, 120: 1, 135: 2, 150:
3, 165: 3, 180: 0, 195: 0, 210: 0, 225: 4, 240: 0, 255: 0, 270: 6, 285: 0}
{10: 22, 20: 46, 30: 130, 40: 140, 50: 45, 60: 19, 70: 9, 80: 0}
[0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 1 0 0
 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0
 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0
 0 0 0 1 0 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0
 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 1 1 1 0 1 1 0 1
 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 1 0 0
 1 0 1 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0
 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 1 0 1 0
 1 0 1 0 1 0 0 1 0 0 0]
```

```
/home/daniyal-ahmed/.local/lib/python3.11/site-
packages/sklearn/cluster/_kmeans.py:1412: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

```
/home/daniyal-ahmed/.local/lib/python3.11/site-
packages/sklearn/cluster/_kmeans.py:1412: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

1.2 Some useful code for the midterm

[10]:
```
import seaborn as sns
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.datasets import fetch_lfw_people
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import GridSearchCV, train_test_split

sns.set()

# Get face data
faces = fetch_lfw_people(min_faces_per_person=60)

# plot face data
fig, ax = plt.subplots(3, 5)
for i, axi in enumerate(ax.flat):
    axi.imshow(faces.images[i], cmap='bone')
    axi.set(xticks=[], yticks=[],
            xlabel=faces.target_names[faces.target[i]])
plt.show()

# split train test set
Xtrain, Xtest, ytrain, ytest = train_test_split(faces.data, faces.target,␣
  ↪random_state=42)

pca = PCA(n_components=150, whiten=True)
svc = SVC(kernel='rbf', class_weight='balanced')
svcpca = make_pipeline(pca, svc)

# Tune model to find best values of C and gamma using cross validation
param_grid = {'svc__C': [1, 5, 10, 50],
              'svc__gamma': [0.0001, 0.0005, 0.001, 0.005]}
kfold = 10
grid = GridSearchCV(svcpca, param_grid, cv=kfold)
grid.fit(Xtrain, ytrain)

print(grid.best_params_)
```

```python
# use the best params explicitly here
pca = PCA(n_components=150, whiten=True)
svc = SVC(kernel='rbf', class_weight='balanced', C=10, gamma=0.005)
svcpca = make_pipeline(pca, svc)

model = BaggingClassifier(svcpca, n_estimators=100).fit(Xtrain, ytrain)
yfit = model.predict(Xtest)

fig, ax = plt.subplots(6, 6)
for i, axi in enumerate(ax.flat):
    axi.imshow(Xtest[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[yfit[i]].split()[-1],
                   color='black' if yfit[i] == ytest[i] else 'red')
fig.suptitle('Predicted Names; Incorrect Labels in Red', size=14)
plt.show()

mat = confusion_matrix(ytest, yfit)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=faces.target_names,
            yticklabels=faces.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.show()

print("Accuracy = ", accuracy_score(ytest, yfit))
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[10], line 1
----> 1 import seaborn as sns
      2 from sklearn.svm import SVC
      3 import matplotlib.pyplot as plt

ModuleNotFoundError: No module named 'seaborn'
```