Worksheet 14

Name: Jeong Yong Yang, Junho Son UID: U95912941, U64222022

Topics

- Naive Bayes
- Model Evaluation

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)

P(Attribute A = Yes | Class = No) = 3/7

P(Attribute B = Divorced | Class = Yes) = 1/3

P(Attribute C = High | Class = No) = 3/7

P(Attribute C = Mid | Class = Yes) = 3/3 = 1

b) Classify the following unseen records:

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

Under Naive Bayes' assumption of independence,

P(Yes, Married, Mid | Class = Yes)

= P(Yes | Class = Yes) * P(Married | Class = Yes) * P(Mid | Class = Yes)

= 0

P(Yes, Married, Mid | Class = No)

= P(Yes | Class = No) * P(Married | Class = No) * P(Mid | Class = No)

= 3/7 * 4/7 * 1/7

= 0.034985

1. (Yes, Married, Mid) belongs to class No because 0.034985 > 0 so we will predict class No

P(No, Divorced, High | Class = Yes)

= P(No | Class = Yes) * P(Divorced | Class = Yes) * P(High | Class = Yes)

= 1 * 1/3 * 0

= 0

P(No, Divorced, High | Class = No)

= P(No | Class = No) * P(Divorced | Class = No) * P(High | Class = No)

= 4/7 * 1/7 * 3/7

= 0.034985

2. (No, Divorced, High) belongs to class No because 0.034985 > 0 so we will predict class No

P(No, Single, High | Class = Yes)

= P(No | Class = Yes) * P(Single | Class = Yes) * P(High | Class = Yes)

= 1 * 2/3 * 0

= 0

P(No, Single, High | Class = No)

= P(No | Class = No) * P(Single | Class = No) * P(High | Class = No)

= 4/7 * 2/7 * 3/7

= 0.069971

3. (No, Single, High) belongs to class No because 0.069971 > 0 so we will predict class No

P(No, Divorced, Low | Class = Yes)

= P(No | Class = Yes) * P(Divorced | Class = Yes) * P(Low | Class = Yes)

= 1 * 1/3 * 0

= 0

P(No, Divorced, Low | Class = No)

= P(No | Class = No) * P(Divorced | Class = No) * P(Low | Class = No)

= 4/7 * 1/7 * 3/7

= 0.034985

4. (No, Divorced, Low) belongs to class No because 0.034985 > 0 so we will predict class No

Model Evaluation

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

In [ ]:
```
actual_class = ["Yes", "No", "No", "Yes", "No", "No", "Yes", "No", "No", "No"]
predicted_class = ["Yes", "No", "Yes", "No", "No", "No", "Yes", "Yes", "Yes", "No"]

def confusion_matrix(actual, predicted):
    trueP = 0
    trueN = 0
    falseP = 0
    falseN = 0
    for x in range(len(actual)):
      if actual[x] == "Yes" and actual[x] == predicted[x]:
        trueP += 1
      elif actual[x] == "No" and actual[x] == predicted[x]:
        trueN += 1
      elif actual[x] == "No" and actual[x] != predicted[x]:
        falseP += 1
      else:
        falseN += 1
    matrix = [[trueP, falseN],[falseP, trueN]]
    return matrix

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

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

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?

The cost is -1(-1) + 5(100) + 10(1) + 0 = 511.

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.

In [ ]:
```
def outputCost(actual, predicted, costMatrix):
  cost = 0
  cost += costMatrix[0][0] * -1
  cost += costMatrix[0][1] * 100
  cost += costMatrix[1][0] * 1
  return cost

print(outputCost(actual_class, predicted_class, confusion_matrix(actual_class, predicted
```

```
101
```

d) Implement functions for the following:

- accuracy
- precision
- recall
- f-measure

and apply them to the above example.

In [ ]:
```
def accuracy(actual, predicted):
  total = len(actual)
  correct = 0
  for x in range(len(actual)):
    if actual[x] == predicted[x]:
      correct += 1
  return correct/total

def precision(actual, predicted):
    trueP = 0
    trueN = 0
    falseP = 0
    falseN = 0
    for x in range(len(actual)):
      if actual[x] == "Yes" and actual[x] == predicted[x]:
        trueP += 1
      elif actual[x] == "No" and actual[x] == predicted[x]:
        trueN += 1
      elif actual[x] == "No" and actual[x] != predicted[x]:
        falseN += 1
      else:
        falseP += 1
    return trueP / (trueP + falseP)

def recall(actual, predicted):
    trueP = 0
    trueN = 0
    falseP = 0
    falseN = 0
    for x in range(len(actual)):
      if actual[x] == "Yes" and actual[x] == predicted[x]:
        trueP += 1
      elif actual[x] == "No" and actual[x] == predicted[x]:
        trueN += 1
      elif actual[x] == "No" and actual[x] != predicted[x]:
        falseN += 1
      else:
        falseP += 1
    return trueP / (trueP + falseN)
```

```
def fMeasure(actual, predicted):
  rec = recall(actual, predicted)
  pre = precision(actual, predicted)
  return 2 * rec * pre / (rec + pre)

print(accuracy(actual_class, predicted_class))
print(precision(actual_class, predicted_class))
print(recall(actual_class, predicted_class))
print(fMeasure(actual_class, predicted_class))
```

```
0.6
0.6666666666666666
0.4
0.5
```

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.

In [ ]:

```
import pandas as pd
import numpy as np
import csv

pdTitanic = pd.read_csv("Titanic.csv")
pdTitanicTwo = pd.read_csv("Titanic2.csv")
df = pd.concat([pdTitanic, pdTitanicTwo], ignore_index=True)
df.head(5)
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0.0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1.0	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1.0	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1.0	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123
4	5	0.0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN

In [ ]:

```
df.isnull().sum()
```

Out[ ]:

```
PassengerId       0
Survived        418
Pclass            0
Name              0
Sex               0
Age             263
SibSp             0
Parch             0
Ticket            0
Fare              1
Cabin          1014
Embarked          2
dtype: int64
```

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  1309 non-null   int64  
 1   Survived     891 non-null    float64
 2   Pclass       1309 non-null   int64  
 3   Name         1309 non-null   object 
 4   Sex          1309 non-null   object 
 5   Age          1046 non-null   float64
 6   SibSp        1309 non-null   int64  
 7   Parch        1309 non-null   int64  
 8   Ticket       1309 non-null   object 
 9   Fare         1308 non-null   float64
 10  Cabin        295 non-null    object 
 11  Embarked     1307 non-null   object 
dtypes: float64(3), int64(4), object(5)
memory usage: 122.8+ KB
```

In [ ]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from collections import Counter

df['has_cabin'] = df['Cabin'].apply(lambda x: 0 if pd.isna(x) else 1)

df['family_members'] = df['SibSp'] + df['Parch']

listName = df['Name'].tolist()
listTitle = []
for x in listName:
  listTitle.append(x.split()[1])

df['title'] = listTitle
title_counts = Counter(listTitle)
threshold = len(listTitle) / 100
results = {}
for item, item_count in title_counts.items():
    results[item] = 1 if item_count > threshold else 0

df['title_type'] = df['title'].map(results)

df.drop(['title'], axis=1, inplace=True)
df = df.dropna(subset=['Fare'])
```

In [ ]:
```
fare_scaled = StandardScaler().fit_transform(df[['Fare']])
kmeans = KMeans(n_clusters=3, random_state=42).fit(fare_scaled)
df['fare_cluster'] = kmeans.labels_
ordered_clusters = df.groupby('fare_cluster')['Fare'].mean().sort_values().index
cluster_mapping = {old: new for new, old in enumerate(ordered_clusters)}
df['fare_price'] = df['fare_cluster'].map(cluster_mapping)

df.drop(['fare_cluster'], axis=1, inplace=True)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: Th
e default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_ini
t` explicitly to suppress the warning
  warnings.warn(
```

In [ ]:
```
df.head(5)
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0.0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1.0	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1.0	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1.0	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123
4	5	0.0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN

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.

In [ ]:

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd

X = df.drop('Survived', axis=1)
y = df['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42

dt = DecisionTreeClassifier()



grid_search = GridSearchCV(estimator=dt, param_grid=param_grid, cv=5, scoring='accuracy'

grid_search.fit(X_train, y_train)

print("Best parameters:", grid_search.best_params_)
print("Best score:", grid_search.best_score_)

predictions = grid_search.predict(X_test)
print("Accuracy on test set:", accuracy_score(y_test, predictions))
print("Classification Report:\n", classification_report(y_test, predictions))
```

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

In [ ]:

```
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
import pandas as pd

X = df.drop('Survived', axis=1)
y = df['Survived']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)

X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.3, random_stat

model = GaussianNB()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

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

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

X = df[['Age', 'Sex']]
y = df['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
train_accuracy_lst = []
test_accuracy_lst = []
for K in range(1, 51):
  knn = KNeighborsClassifier(n_neighbors=K)
  knn.fit(X_train, y_train)
  y_train_pred = knn.predict(X_train)
  y_test_pred = knn.predict(X_test)
  train_accuracy = accuracy_score(y_train, y_train_pred)
  test_accuracy = accuracy_score(y_test, y_test_pred)
  train_accuracy_lst.append(train_accuracy)
  test_accuracy_lst.append(test_accuracy)
plt.figure(figsize=(12, 8))
plt.plot(range(1, 51), train_accuracy_lst, label='Training Accuracy')
plt.plot(range(1, 51), test_accuracy_lst, label='Testing Accuracy')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.title('Training and Testing Accuracy for Different K Values')
plt.legend()
plt.xticks(range(1, 51))
plt.show()
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
y_train_pred = knn.predict(X_train)
```

```
y_test_pred = knn.predict(X_test)
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
print(train_accuracy, test_accuracy)
```

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

https://www.kaggle.com/competitions/titanic/submissions

Some useful code for the midterm

In [ ]:
```
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=4

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_)

# 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)
```

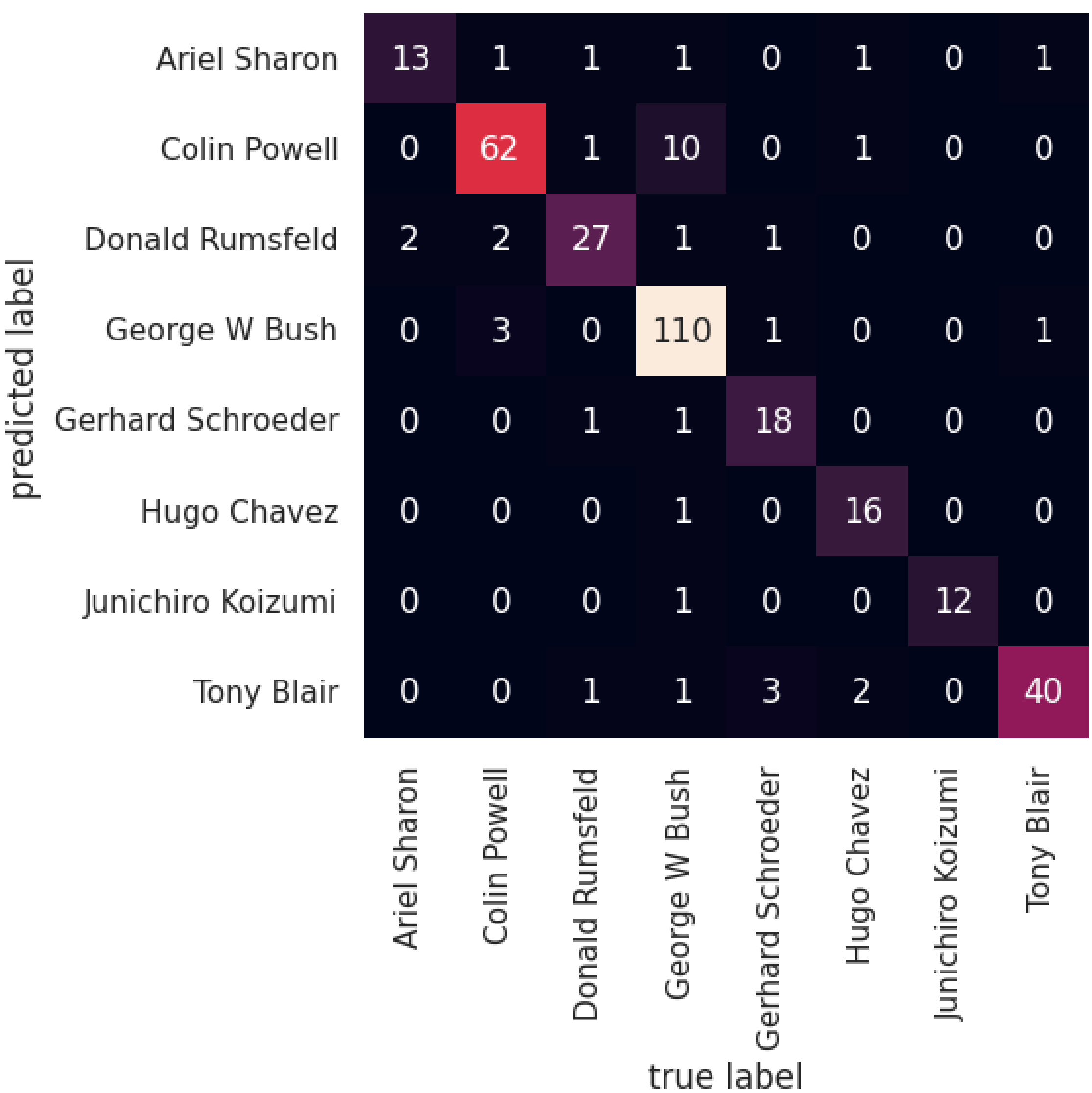
```python
fig, ax = plt.subplots(6, 6)
for i, axi in enumerate(ax.flat):
    axi.imshow(Xtest[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[yfit[i]].split()[-1],
                   color='black' if yfit[i] == ytest[i] else 'red')
fig.suptitle('Predicted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


```
{'svc__C': 10, 'svc__gamma': 0.005}
```

Predicted Names; Incorrect Labels in Red




```
Accuracy =  0.884272997032641
```