Worksheet 14

Name: Ziye Chen
UID: U98411098

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)

(Yes, Married, Mid)

- P(A = Yes | class = Yes) * P(B = Married | class = Yes) * P(C = Mid| class = Yes) = 0 / 3 * 0 / 3 * 3 / 3 = 0
- P(A = Yes | class = No) * P(B = Married | class = No) * P(C = Mid| class = No) = 3 / 7 * 4 / 7 * 1 / 7 = 12 / 343 = 0.03498
- Class = NO

(No, Divorced, High)

- P(A = No | class = Yes) * P(B = Divorced | class = Yes) * P(C = High| class = Yes) = 3/3 * 1/3 * 0/3 = 0
- P(A = No | class = No) * P(B = Divorced | class = No) * P(C = High| class = No) = 4/7 * 1/7 * 3/7 = 12 / 343 = 0.03498
- Class = No

(No, Single, High)

- P(A = No | class = Yes) * P(B = Single | class = Yes) * P(C = High| class = Yes) = 3/3 * 2/3 * 0/3 = 0
- P(A = No | class = No) * P(B = Single | class = No) * P(C = High| class = No) = 4/7 * 2/7 * 3/7 = 24 / 343 = 0.06996
- Class = No

(No, Divorced, Low)

- P(A = No | class = Yes) * P(B = Divorced | class = Yes) * P(C = Low| class = Yes) = 3/3 * 1/3 * 0/3 = 0
- P(A = No | class = No) * P(B = Divorced | class = No) * P(C = Low| class = No) = 4/7 * 1/7 * 3/7 = 12/343 = 0.03498
- 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 [1]:
```
actual_class = ["Yes", "No", "No", "Yes", "No", "No", "Yes", "No", "No", "No
predicted_class = ["Yes", "No", "Yes", "No", "No", "No", "Yes", "Yes", "Yes"

def confusion_matrix(actual, predicted):
    tp, fp, fn, tn = 0, 0, 0, 0

    n = len(actual_class)
    for i in range(n):
        if predicted[i]=='Yes':
            if actual[i]=='Yes':
                tp+=1
            elif actual[i]=='No':
                fp+=1
        elif predicted[i]=='No':
            if actual[i]=='Yes':
                fn+=1
            elif actual[i]=='No':
                tn+=1

    return f"{tp}  {fn} \n{fp}  {tn}"

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?

2 * (-1) + 1 * 5 + 3 * 10 + 4 * 0 = 33

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 [2]:
```
def test(actual, predicted, cost):
    tp, fp, fn, tn = 0, 0, 0, 0
    ctp, cfn, cfp, ctn = cost
    n = len(actual_class)
```

```
    for i in range(n):
        if predicted[i]=='Yes':
            if actual[i]=='Yes':
                tp+=1
            elif actual[i]=='No':
                fp+=1
        elif predicted[i]=='No':
            if actual[i]=='Yes':
                fn+=1
            elif actual[i]=='No':
                tn+=1

    return tp*ctp + fn*cfn + fp*cfp + tn*ctn

cost1 = [-1, 5, 10, 0]
print(test(actual_class, predicted_class, cost1))
```

33

d) Implement functions for the following:

- accuracy
- precision
- recall
- f-measure

and apply them to the above example.

In [3]:
```
def accu(actual, predicted):
    tp, fp, fn, tn = 0, 0, 0, 0

    n = len(actual_class)
    for i in range(n):
        if predicted[i]=='Yes':
            if actual[i]=='Yes':
                tp+=1
            elif actual[i]=='No':
                fp+=1
        elif predicted[i]=='No':
            if actual[i]=='Yes':
                fn+=1
            elif actual[i]=='No':
                tn+=1

    print(f"{tp}  {fn} \n{fp}  {tn}")

    return (tp + tn) / sum([tp, fp, fn, tn])

print(f"{accu(actual_class, predicted_class)*100}%")
```

```
2   1
3   4
60.0%
```

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 [ ]:
```python
def extract_title(name):
    return name.split(',')[1].split('.')[0].strip()

def age_group(age):
    if age < 12:
        return 0  # child
    elif age < 60:
        return 1  # adult
    else:
        return 2  # senior

def extract_deck(cabin):
```

```python
    if pd.isna(cabin):
        return -1  # Unknown
    return ord(cabin[0]) - ord('A')  # Convert letter to numerical value
```

In [ ]:
```python
data = [train_ds, test_ds]
for dataset in data:
    dataset['Fare'] = (dataset['Fare'].fillna(0)).astype(int)

data = [train_ds, test_ds]
for dataset in data:
    dataset['has_cabin'] = dataset['Cabin'].apply(lambda x: 0 if pd.isna(x)

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

    dataset['Title'] = dataset['Name'].apply(extract_title)
    title_counts = dataset['Title'].value_counts()
    threshold = 10  # Define 'common' as a title appearing more than 10 time
    common_titles = title_counts[title_counts > threshold].index.tolist()
    dataset['title_type'] = dataset['Title'].apply(lambda x: 1 if x in commo

    kmeans = KMeans(n_clusters=3, random_state=0).fit(dataset[['Fare']])
    dataset['fare_price'] = kmeans.labels_

    dataset['age_group'] = dataset['Age'].apply(age_group)

    dataset['deck_level'] = dataset['Cabin'].apply(extract_deck)
```

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 [ ]:
```python
dtree = DecisionTreeClassifier(random_state=42)

# Set up the parameters grid for tuning
param_grid = {
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 10],
    'criterion': ['gini', 'entropy','gini']
}

# Use grid search for parameter tuning with cross-validation
grid_search = GridSearchCV(dtree, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

```
# Get the best model
best_tree = grid_search.best_estimator_

# Evaluate the model on the test set
y_pred = best_tree.predict(X_test)
print(classification_report(y_test, y_pred))
```

By using cross-validation and keeping the test set separate, we ensure that the tuning process is robust and that we are not overfitting to the training set.

jupyter

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

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report

# Reduce the dimension by dropping features
X = df.drop('Survived', axis=1)  # Features
y = df['Survived']  # Target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# Dimensionality Reduction with PCA
pca = PCA(n_components=15)  # keep 15 variances
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Naive Bayes Classifier
nb = GaussianNB()
nb.fit(X_train_pca, y_train)

# Evaluate the model
y_pred = nb.predict(X_test_pca)
print(classification_report(y_test, y_pred))
```

jupyter

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

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report

# Initialize the individual models
knn = KNeighborsClassifier(n_neighbors=5)
dtree = DecisionTreeClassifier(max_depth=None)
nb = GaussianNB()

# Create an ensemble classifier
ensemble = VotingClassifier(estimators=[
    ('knn', knn),
    ('dtree', dtree),
    ('nb', nb)
], voting='hard')

# Train the ensemble classifier
ensemble.fit(X_train, y_train)

# Evaluate the ensemble classifier
y_pred = ensemble.predict(X_test)
print(classification_report(y_test, y_pred))
```

jupyter

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/code/ziechan/cs506midterm?scriptVersionId=168042530

Some useful code for the midterm

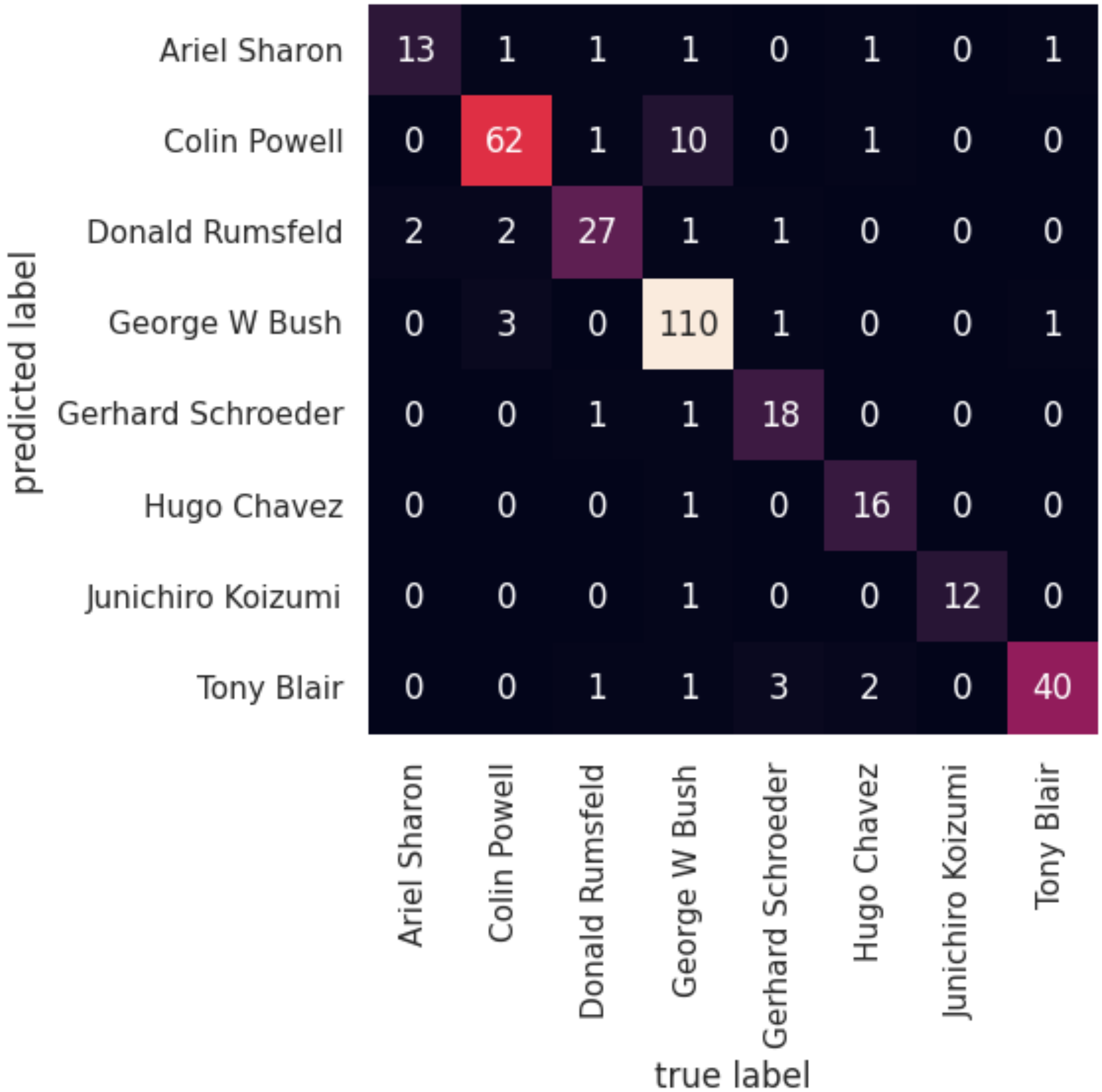
In [1]:
```
import seaborn as sns
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.datasets import fetch_lfw_people
from sklearn.ensemble 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, ra

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)

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
```