worksheet_14

March 23, 2024

1 Worksheet 14

Name: **Bowen Li**
UID: **U79057147**

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)

$$P(A = \text{Yes} \mid \text{Class} = \text{No}) = \frac{P(A = \text{Yes, Class} = \text{No})}{P(\text{Class} = \text{No})} = \frac{3/10}{7/10} = \boxed{\frac{3}{7}}$$

$$P(B = \text{Divorced} \mid \text{Class} = \text{Yes}) = \frac{P(B = \text{Divorced, Class} = \text{Yes})}{P(\text{Class} = \text{Yes})} = \frac{1/10}{3/10} = \boxed{\frac{1}{3}}$$

$$P(C = \text{High} \mid \text{Class} = \text{No}) = \frac{P(C = \text{High}, \text{Class} = \text{No})}{P(\text{Class} = \text{No})} = \frac{3/10}{7/10} = \boxed{\frac{3}{7}}$$

$$P(C = \text{Mid} \mid \text{Class} = \text{Yes}) = \frac{P(C = \text{Mid}, \text{Class} = \text{Yes})}{P(\text{Class} = \text{Yes})} = \frac{3/10}{3/10} = \boxed{1}$$

b) Classify the following unseen records:

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

- (Yes, Married, Mid)

$$P(\text{Class} = \text{Yes}|\text{Yes, Married, Mid}) \propto P(\text{Yes, Married, Mid}|\text{Class} = \text{Yes}) = P(\text{Yes}|\text{Class} = \text{Yes})P(\text{Married}|\text{Cl}$$

$$= 0 \cdot 0 \cdot 1 = 0$$

$$P(\text{Class} = \text{No}|\text{Yes, Married, Mid}) \propto P(\text{Yes, Married, Mid}|\text{Class} = \text{No}) = P(\text{Yes}|\text{Class} = \text{No})P(\text{Married}|\text{Clas}$$

$$= \frac{3}{7} \cdot \frac{4}{7} \cdot \frac{1}{7} > 0$$

$P(\text{Class} = \text{Yes}|\text{Yes, Married, Mid}) < P(\text{Class} = \text{No}|\text{Yes, Married, Mid})$; so the class is $\boxed{\text{No.}}$

- (No, Divorced, High)

$$P(\text{Class} = \text{Yes}|\text{No, Divorced, High}) \propto P(\text{No, Divorced, High}|\text{Class} = \text{Yes}) = P(\text{No}|\text{Class} = \text{Yes})P(\text{Divorced}|$$

$$= 1 \cdot \frac{1}{3} \cdot 0 = 0$$

$$P(\text{Class} = \text{No}|\text{No, Divorced, High}) \propto P(\text{No, Divorced, High}|\text{Class} = \text{No}) = P(\text{No}|\text{Class} = \text{No})P(\text{Divorced}|\text{C}$$

$$= \frac{4}{7} \cdot \frac{1}{7} \cdot \frac{3}{7} > 0$$

$P(\text{Class} = \text{Yes}|\text{No, Divorced, High}) < P(\text{Class} = \text{No}|\text{No, Divorced, High})$; so the class is $\boxed{\text{No.}}$

- (No, Single, High)

$$P(\text{Class} = \text{Yes}|\text{No, Single, High}) \propto P(\text{No, Single, High}|\text{Class} = \text{Yes}) = P(\text{No}|\text{Class} = \text{Yes})P(\text{Single}|\text{Class} =$$

$$= 1 \cdot \frac{2}{3} \cdot 0 = 0$$

$$P(\text{Class} = \text{No}|\text{No, Single, High}) \propto P(\text{No, Single, High}|\text{Class} = \text{No}) = P(\text{No}|\text{Class} = \text{No})P(\text{Single}|\text{Class} = \text{No}$$

$$= \frac{4}{7} \cdot \frac{2}{7} \cdot \frac{3}{7} > 0$$

$P(\text{Class} = \text{Yes}|\text{No, Single, High}) < P(\text{Class} = \text{No}|\text{No, Single, High})$; so the class is $\boxed{\text{No.}}$

- (No, Divorced, Low)

  $P(\text{Class} = \text{Yes}|\text{No, Divorced, Low}) \propto P(\text{No, Divorced, Low}|\text{Class} = \text{Yes}) = P(\text{No}|\text{Class} = \text{Yes})P(\text{Divorced}|\text{C}$

  $$= 1 \cdot \frac{1}{3} \cdot 0 = 0$$

  $P(\text{Class} = \text{No}|\text{No, Divorced, Low}) \propto P(\text{No, Divorced, Low}|\text{Class} = \text{No}) = P(\text{No}|\text{Class} = \text{No})P(\text{Divorced}|\text{Cla}$

  $$= \frac{4}{7} \cdot \frac{1}{7} \cdot \frac{3}{7} > 0$$

  $P(\text{Class} = \text{Yes}|\text{No, Divorced, Low}) < P(\text{Class} = \text{No}|\text{No, Divorced, Low})$; so the class is $\boxed{\text{No.}}$

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

[1]:
```python
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):
    n = len(actual)
    if n != len(predicted):
        raise ValueError("Predictions don't match label length.")

    confusion_mat = [[0,0],[0,0]]

    for i in range(n):
        if actual_class[i] == "Yes" and predicted_class[i] == "Yes":
            confusion_mat[0][0] += 1
        elif actual_class[i] == "Yes" and predicted_class[i] == "No":
            confusion_mat[0][1] += 1
        elif actual_class[i] == "No" and predicted_class[i] == "Yes":
            confusion_mat[1][0] += 1
        else:
            confusion_mat[1][1] += 1
    return confusion_mat

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?

$$2(-1) + 1(5) + 3(10) + 4(0) = \boxed{33}$$

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.

[2]:
```
def compute_cost(actual, predicted, cost_matrix):
    confusion_mat = confusion_matrix(actual, predicted)

    n = len(cost_matrix)
    m = len(cost_matrix[0])

    if n != len(confusion_mat) or m != len(confusion_mat[0]):
        raise ValueError("Dimensions of cost matrix don't match confusion␣
 ↪matrix.")

    cost = 0
    for i in range(n):
        for j in range(m):
            cost += cost_matrix[i][j] * confusion_mat[i][j]

    return cost

cost_mat = [[-1,5],[10,0]]

print(compute_cost(actual_class, predicted_class, cost_mat))
```

```
33
```

d) Implement functions for the following:

- accuracy
- precision
- recall
- f-measure

and apply them to the above example.

[3]:
```
def accuracy(actual, predicted):
    confusion_mat = confusion_matrix(actual, predicted)
    num_correct = 0
    for i in range(len(confusion_mat)):
        num_correct += confusion_mat[i][i]

    total = 0
    for row in confusion_mat:
        total += sum(row)
```

```python
    return num_correct / total

print("Accuracy:", accuracy(actual_class, predicted_class))

def precision(actual, predicted):
    confusion_mat = confusion_matrix(actual, predicted)
    return confusion_mat[0][0] / (confusion_mat[0][0] + confusion_mat[1][0])

print("Precision:", precision(actual_class, predicted_class))

def recall(actual, predicted):
    confusion_mat = confusion_matrix(actual, predicted)
    return confusion_mat[0][0] / (confusion_mat[0][0] + confusion_mat[0][1])

print("Recall:", recall(actual_class, predicted_class))

def f_measure(actual, predicted):
    p = precision(actual, predicted)
    r = recall(actual, predicted)

    return 2 * p * r / (p + r)

print("F-measure:", f_measure(actual_class, predicted_class))
```

```
Accuracy: 0.6
Precision: 0.4
Recall: 0.6666666666666666
F-measure: 0.5
```

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.

Kaggle Username: **lib250**

1.1.1 Get Imports and Load Data

[4]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import LeaveOneOut, cross_val_score
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.cluster import KMeans
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier, HistGradientBoostingClassifier

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
```

[5]:
```
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')

print(f"Training Set: {train_df.shape}")
print(f"Testing Set: {test_df.shape}")
```

```
Training Set: (891, 12)
Testing Set: (418, 11)
```

[6]:
```
train_df.head()
```

[6]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0
```

```
   Parch             Ticket     Fare Cabin Embarked
0      0          A/5 21171   7.2500   NaN        S
1      0           PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0             113803  53.1000  C123        S
4      0             373450   8.0500   NaN        S
```

1.1.2 Extract Features

Here we extract the numerical `has_cabin`, `family_members`, `title_type`, and `fare_type` features as described in the instructions.

We additionally extract `is_child` as an indicator where `1` indicates a person `18` or under and `0` indicates an adult. This could be useful since children have priority in terms of lifeboat access. We extract `is_woman` for the same reason.

We also enumerate the `deck` as the first letter of the cabin number indicates where physically on the ship the person's cabin is located. Spatial information could be useful as proximity to danger and safe zones on the ship could affect survival rate.

Extract Title Here, we get all the titles in the dataset and count how frequently they appear. We consider the 3 most frequent titles, `Mr.`, `Miss.` and `Mrs.` to be `common` titles while the rest `Master.` onwards to be uncommon, since there's a big dip in frequency after `Mrs.`.

[7]:
```python
def extract_title(name):
  strs = name.split()
  for s in strs:
    if '.' in s:
      return s

titles = train_df["Name"].copy().apply(extract_title)
```

[8]:
```python
titles.value_counts()
```

[8]:
```
Mr.          517
Miss.        182
Mrs.         125
Master.       40
Dr.            7
Rev.           6
Mlle.          2
Major.         2
Col.           2
Countess.      1
Capt.          1
Ms.            1
Sir.           1
Lady.          1
Mme.           1
```

```
Don.            1
Jonkheer.       1
Name: Name, dtype: int64
```

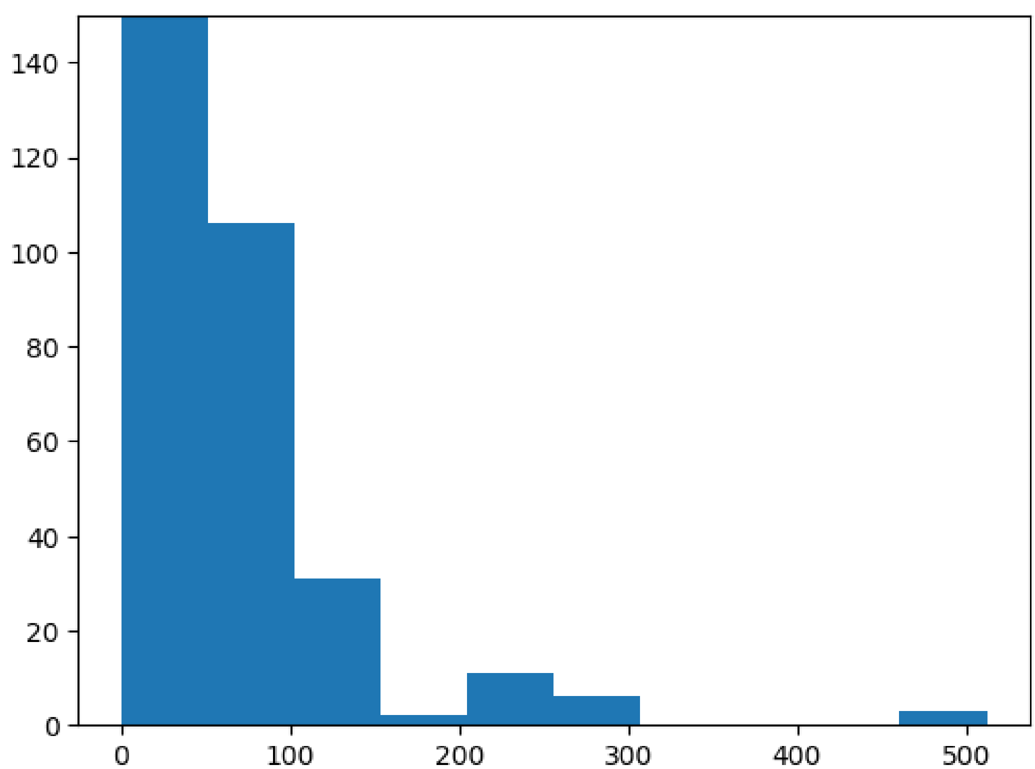
[9]:
```python
common_titles = ['Mr.', 'Miss.', 'Mrs.', 'Ms.']
```

[10]:
```python
def has_common_title(name):
  title = extract_title(name)
  if title in common_titles:
    return 1
  return 0
```

Extract Fare Type Here we perform K-means clustering to bucket the fares. We first plot a histogram of all the fares and can see that they roughly concentrate into 3 groups. We have a local peak at around 0-50, 200-250, and the last bin at around 500 (The value of the first bin is much larger than the bounds of the graph, we limit the y-axis to better see the smaller values).

As such we can use K-means to collect the fares into 3 buckets, potentially representing a lower, middle, and upper economic class.

[11]:
```python
plt.hist(train_df["Fare"])
plt.ylim(0,150)
plt.show()
```



[12]:
```
fare_array = train_df["Fare"].to_numpy().reshape((-1, 1))

fare_kmeans = KMeans(n_clusters=3, random_state=0, n_init="auto").
↪fit(fare_array)
```

[13]:
```
fare_kmeans.cluster_centers_
```

[13]:
```
array([[ 82.92256875],
       [279.308545  ],
       [ 15.3602868 ]])
```

[14]:
```
sorted_idx = np.argsort(fare_kmeans.cluster_centers_.flatten())
ordered_fare_centers = np.zeros(3, dtype=int)
ordered_fare_centers[sorted_idx] = np.arange(3)
ordered_fare_centers
```

[14]:
```
array([1, 2, 0])
```

Process Data and Extract Features

[15]:
```
def extract_features(df):
  y = None
  if "Survived" in df.columns:
    y = df["Survived"].to_numpy().copy()

  df["Age"].fillna(df["Age"].mean(), inplace=True)
  df["Fare"].fillna(df["Fare"].mean(), inplace=True)

  df["is_child"] = (df["Age"] < 19).astype(int)
  df["is_woman"] = (df["Sex"].str.lower() == "female").astype(int)
  df["has_cabin"] = df["Cabin"].notna().astype(int)
  df["deck"] = (df["Cabin"].str[:1])
  df["deck"].fillna('N', inplace=True)
  df["family_members"] = df["SibSp"] + df["Parch"]
  df["title_type"] = df["Name"].apply(has_common_title)

  deck_to_num = {'N': 0, 'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6, 'G':␣
↪7, 'T': 8}
  df["deck"] = df["deck"].apply(lambda letter: deck_to_num[letter])

  fare_np = df["Fare"].to_numpy().reshape((-1, 1))
  fare_buckets_np = fare_kmeans.predict(fare_np)
  df["fare_type"] = pd.Series(fare_buckets_np).apply(lambda c:␣
↪ordered_fare_centers[c])
```

```
  features = df[["Pclass", "has_cabin", "is_child", "is_woman", "deck",␣
↪"family_members", "title_type", "fare_type"]].copy()


  return features.to_numpy(), y
```

[16]:
```
X_train, y_train = extract_features(train_df)
X_test, y_test = extract_features(test_df)
```

1.1.3 Decision Tree

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.

The model is likely not overfitting to the training set as it performs relatively well on leave-one-out cross validation runs. We also select the minimum max_depth which gets us decent performance, which reduces the chances of overfitting.

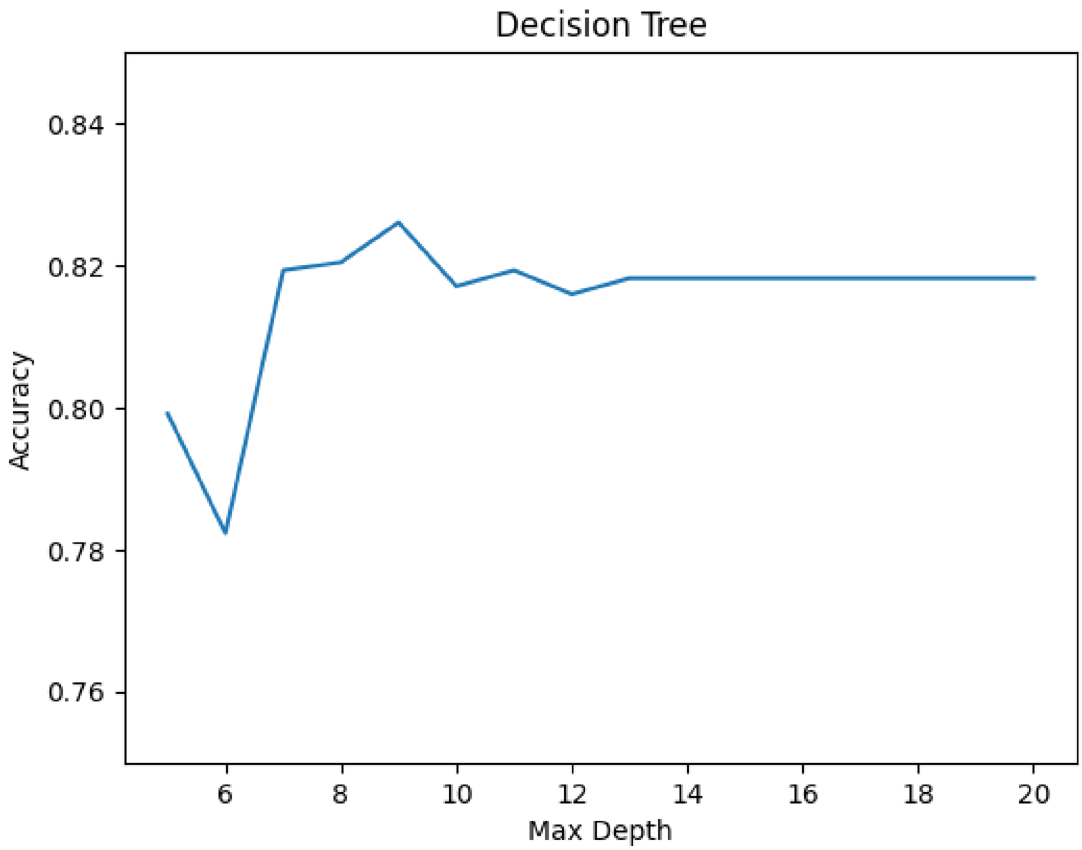
[17]:
```
def cross_val_acc(model, X, y):
  splits = LeaveOneOut()

  results = cross_val_score(model, X, y, cv=splits)
  return results.sum() / results.shape[0]

dt_test = DecisionTreeClassifier(random_state=0)
cross_val_acc(dt_test, X_train, y_train)
```

[17]: 0.8181818181818182

[18]:
```
upper_max_depth = 20
lower_max_depth = 5

max_depth_vals = np.arange(lower_max_depth, upper_max_depth+1)

def apply_max_depth_DT(max_depth):
  model = DecisionTreeClassifier(random_state=0, max_depth=max_depth)
  return cross_val_acc(model, X_train, y_train)

vec_apply_depth = np.vectorize(apply_max_depth_DT)
```

[19]:
```
md_acc = vec_apply_depth(max_depth_vals)
```

[20]:
```
plt.plot(max_depth_vals, md_acc)
plt.ylim(0.75, 0.85)
plt.title("Decision Tree")
plt.ylabel("Accuracy")
plt.xlabel("Max Depth")
plt.show()
```

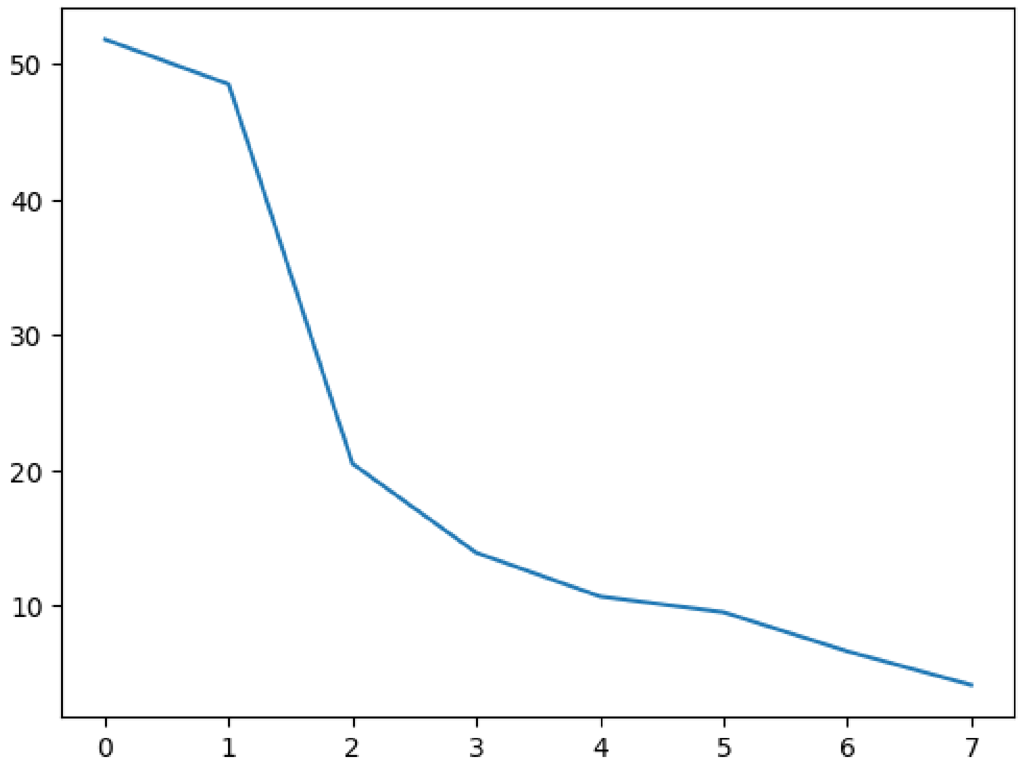


1.1.4 PCA and Naive Bayes

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

We'll first check the singular values to get a sense of how the information gain falls off with each dimension. It seems to taper off at around 3-4 components.

[21]:
```
pca_full = PCA(n_components=8)
pca_full.fit(X_train)

plt.plot(pca_full.singular_values_)
plt.show()
```



[22]:
```
dim_reduce = PCA(n_components=4)
```

We'll initialize the Naive Bayes model with the priors since historically the Titanic had a recorded 37% survival rate.

[23]:
```
naive_bayes = GaussianNB(priors=[.63,.37])
cross_val_acc(naive_bayes, X_train, y_train)
```

[23]: 0.7306397306397306

[24]:
```
pca_nb = make_pipeline(dim_reduce, naive_bayes)
cross_val_acc(pca_nb, X_train, y_train)
```

[24]: 0.7755331088664422

Next we test if scaling helps, which it doesn't seem to.

[25]:
```
scaler = StandardScaler()
standardized_pca_nb = make_pipeline(scaler, dim_reduce, naive_bayes)
cross_val_acc(standardized_pca_nb, X_train, y_train)
```

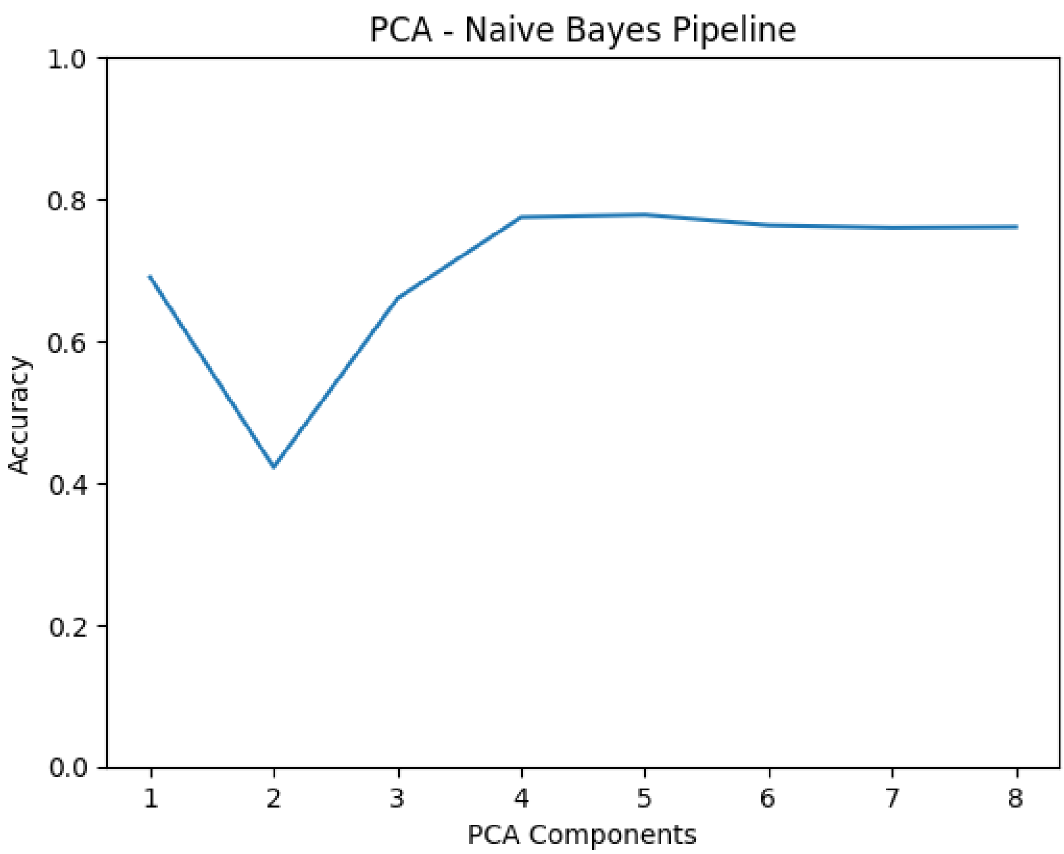
[25]: 0.7138047138047138

Visualize how Naive Bayes performs as we change the number of PCA components.

[26]:
```
test_n_components = np.arange(1, 9)

def apply_pca_n(num_components):
  pca = PCA(n_components=num_components)
  model = make_pipeline(pca, GaussianNB(priors=[.37,.63]))
  return cross_val_acc(model, X_train, y_train)

vec_pca_n = np.vectorize(apply_pca_n)
```

[27]:
```
pca_n_accuracies = vec_pca_n(test_n_components)
```

[28]:
```
plt.plot(test_n_components, pca_n_accuracies)
plt.ylim(0, 1)
plt.title("PCA - Naive Bayes Pipeline")
plt.ylabel("Accuracy")
plt.xlabel("PCA Components")
plt.show()
```



1.1.5 Ensemble

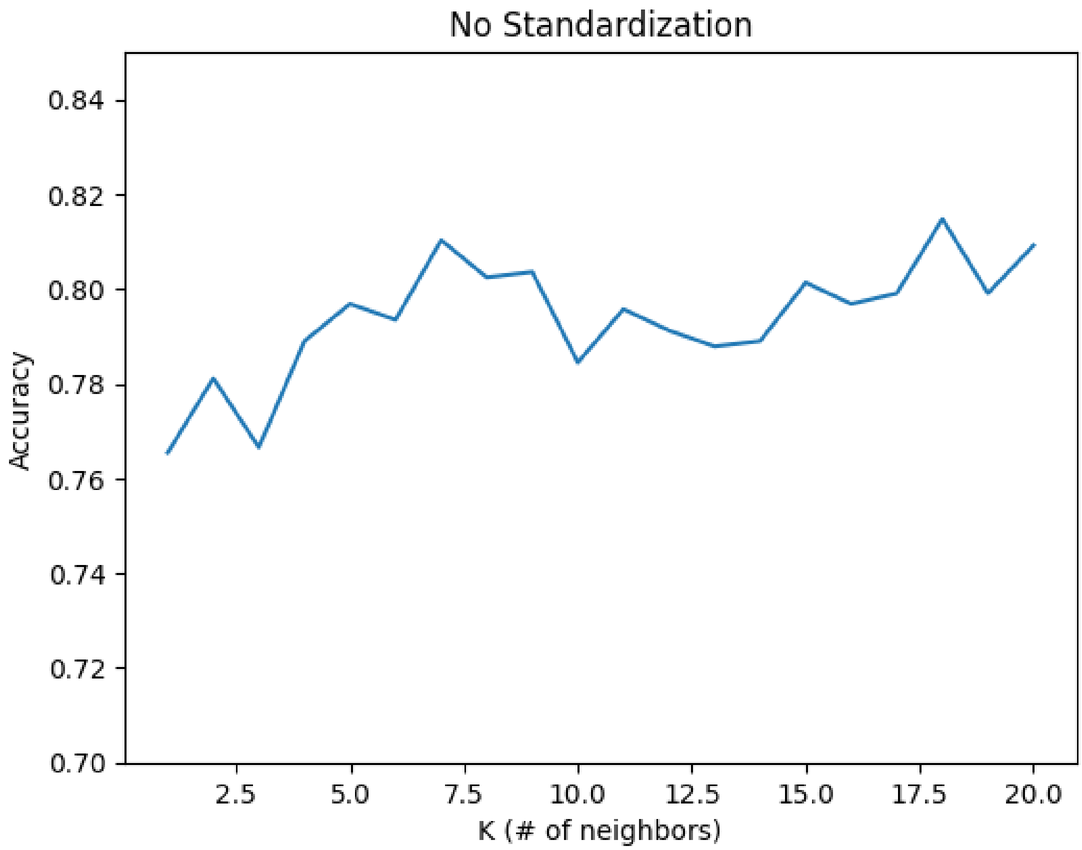
d) Create an ensemble classifier using a combination of KNN, Decision Trees, and Naive Bayes models. Evaluate this classifier.

First let's search for the best KNN for this feature set. We test different values of `k` and different ways to standardize the features of the data.

[29]:
```
def apply_knn_no_stand(k):
  model = KNeighborsClassifier(n_neighbors=k)
  return cross_val_acc(model, X_train, y_train)

def apply_knn_stand(k):
  knn = KNeighborsClassifier(n_neighbors=k)
  model = make_pipeline(StandardScaler(), knn)
  return cross_val_acc(model, X_train, y_train)

def apply_knn_minmax(k):
  knn = KNeighborsClassifier(n_neighbors=k)
  model = make_pipeline(MinMaxScaler(), knn)
  return cross_val_acc(model, X_train, y_train)

vec_knn_no_stand = np.vectorize(apply_knn_no_stand)
vec_knn_stand = np.vectorize(apply_knn_stand)
vec_knn_minmax = np.vectorize(apply_knn_minmax)

k_vals = np.arange(1, 21)
```

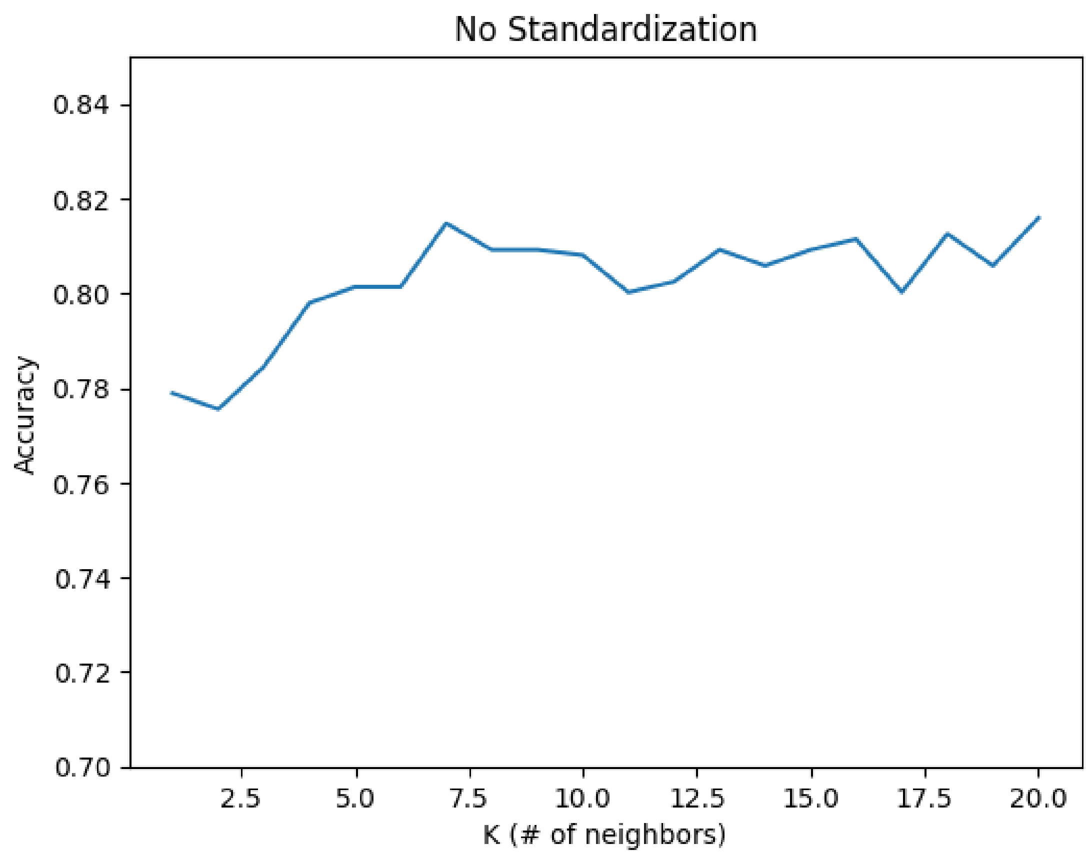
[30]:
```
knn_acc_none = vec_knn_no_stand(k_vals)
```

[31]:
```
plt.plot(k_vals, knn_acc_none)
plt.ylim(0.7, 0.85)
plt.title("No Standardization")
plt.ylabel("Accuracy")
plt.xlabel("K (# of neighbors)")
plt.show()
```



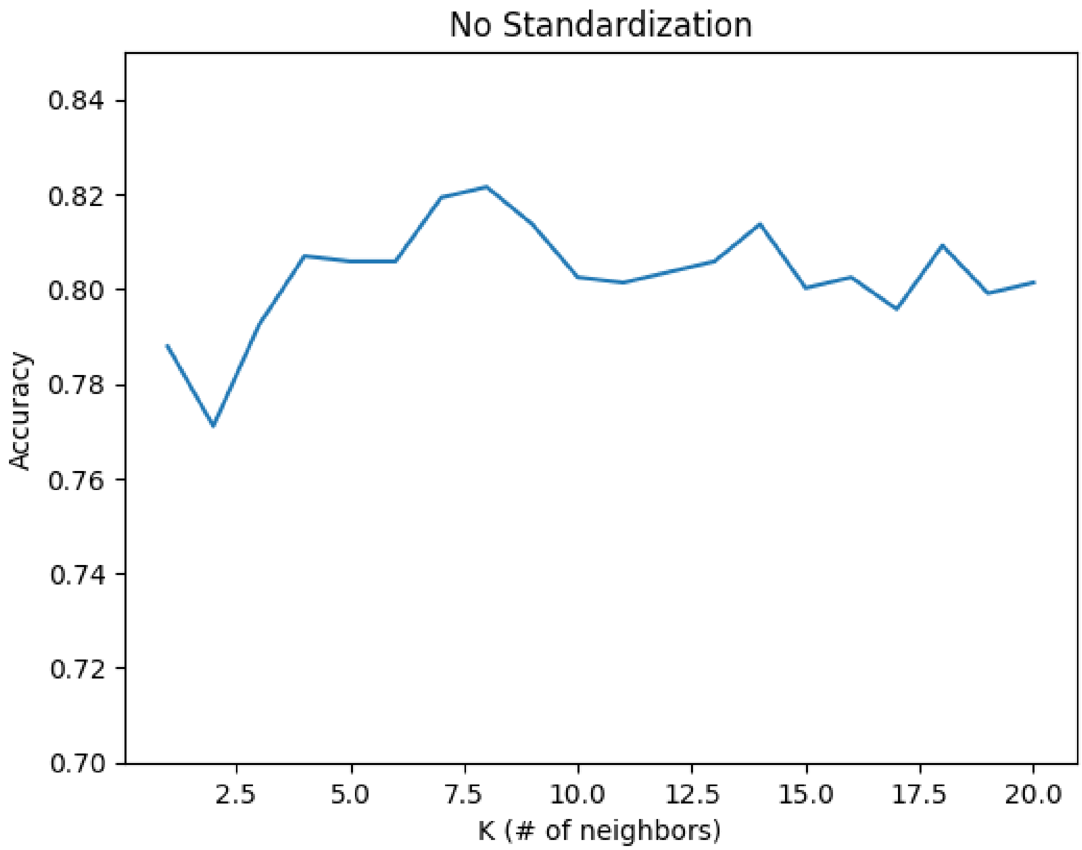
[32]:
```
knn_acc_stand = vec_knn_stand(k_vals)
```

[33]:
```
plt.plot(k_vals, knn_acc_stand)
plt.ylim(0.7, 0.85)
plt.title("No Standardization")
plt.ylabel("Accuracy")
plt.xlabel("K (# of neighbors)")
plt.show()
```



[34]:
```
knn_acc_minmax = vec_knn_minmax(k_vals)
```

[35]:
```
plt.plot(k_vals, knn_acc_minmax)
plt.ylim(0.7, 0.85)
plt.title("No Standardization")
plt.ylabel("Accuracy")
plt.xlabel("K (# of neighbors)")
plt.show()
```



Optimal here is 8 neighbors with `MinMax` standardization.

Check if PCA can help (doesn't seem like it).

[36]:
```
pca_knn_test = make_pipeline(MinMaxScaler(),
                             PCA(n_components=4),
                             KNeighborsClassifier(n_neighbors=15))
cross_val_acc(pca_knn_test, X_train, y_train)
```

[36]: 0.7946127946127947

[37]:
```
best_dt = DecisionTreeClassifier(random_state=0, max_depth=9)
best_nb = make_pipeline(PCA(n_components=4), GaussianNB(priors=[.63, .37]))
best_knn = make_pipeline(MinMaxScaler(), KNeighborsClassifier(n_neighbors=8))

ensemble_model = VotingClassifier(
    estimators=[('dt', best_dt), ('nb', best_nb), ('knn', best_knn)],
    voting='hard'
)
```

[38]:
```
for model, name in zip([best_dt, best_nb, best_knn, ensemble_model], ['Decision␣
↪Tree', 'Naive Bayes', 'KNN', 'Ensemble']):
    scores = cross_val_acc(model, X_train, y_train)
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(),␣
↪name))
```

```
Accuracy: 0.83 (+/- 0.00) [Decision Tree]
Accuracy: 0.78 (+/- 0.00) [Naive Bayes]
Accuracy: 0.82 (+/- 0.00) [KNN]
Accuracy: 0.82 (+/- 0.00) [Ensemble]
```

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

It seems the best model was still the decision tree, so we'll go with that.

[39]:
```
best_model = DecisionTreeClassifier(random_state=0, max_depth=9)
best_model.fit(X_train, y_train)
```

[39]:
```
DecisionTreeClassifier(max_depth=9, random_state=0)
```

[40]:
```
predictions = best_model.predict(X_test)
```

[41]:
```
submission_df = test_df[["PassengerId"]].copy()
submission_df["Survived"] = predictions
submission_df
```

[41]:

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	0
..	…	…
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	1

[418 rows x 2 columns]

[42]:
```
submission_df.to_csv('titanic_submission.csv', index=False)
```

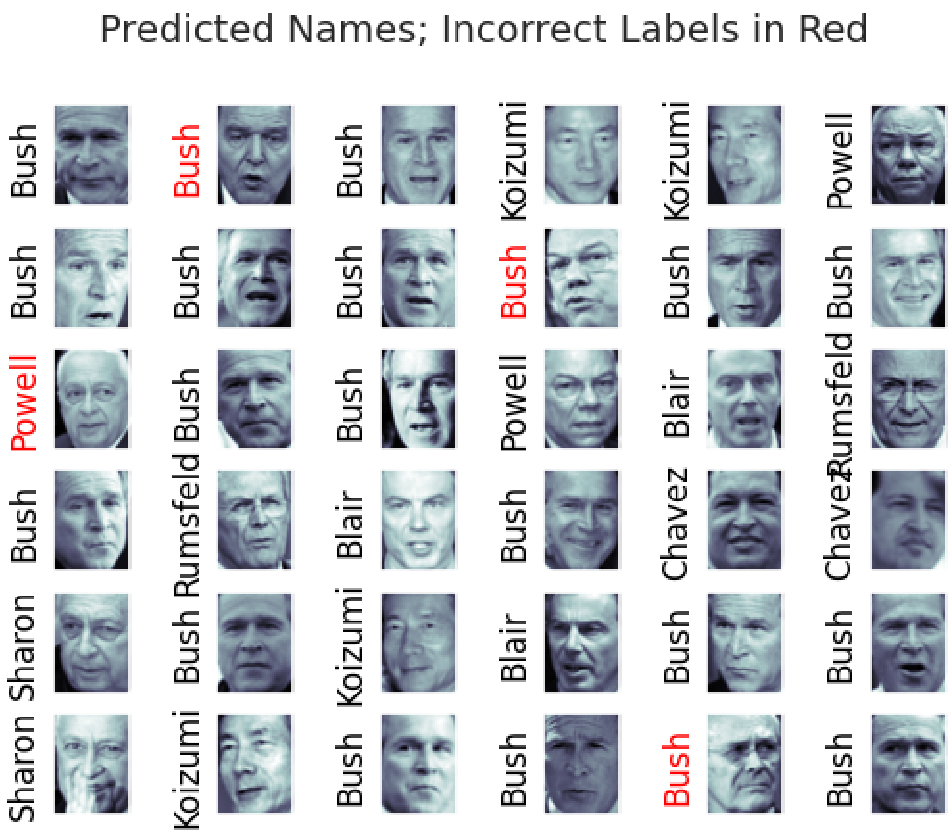
The predictions scored `0.76315` on Kaggle under the username `lib250`.

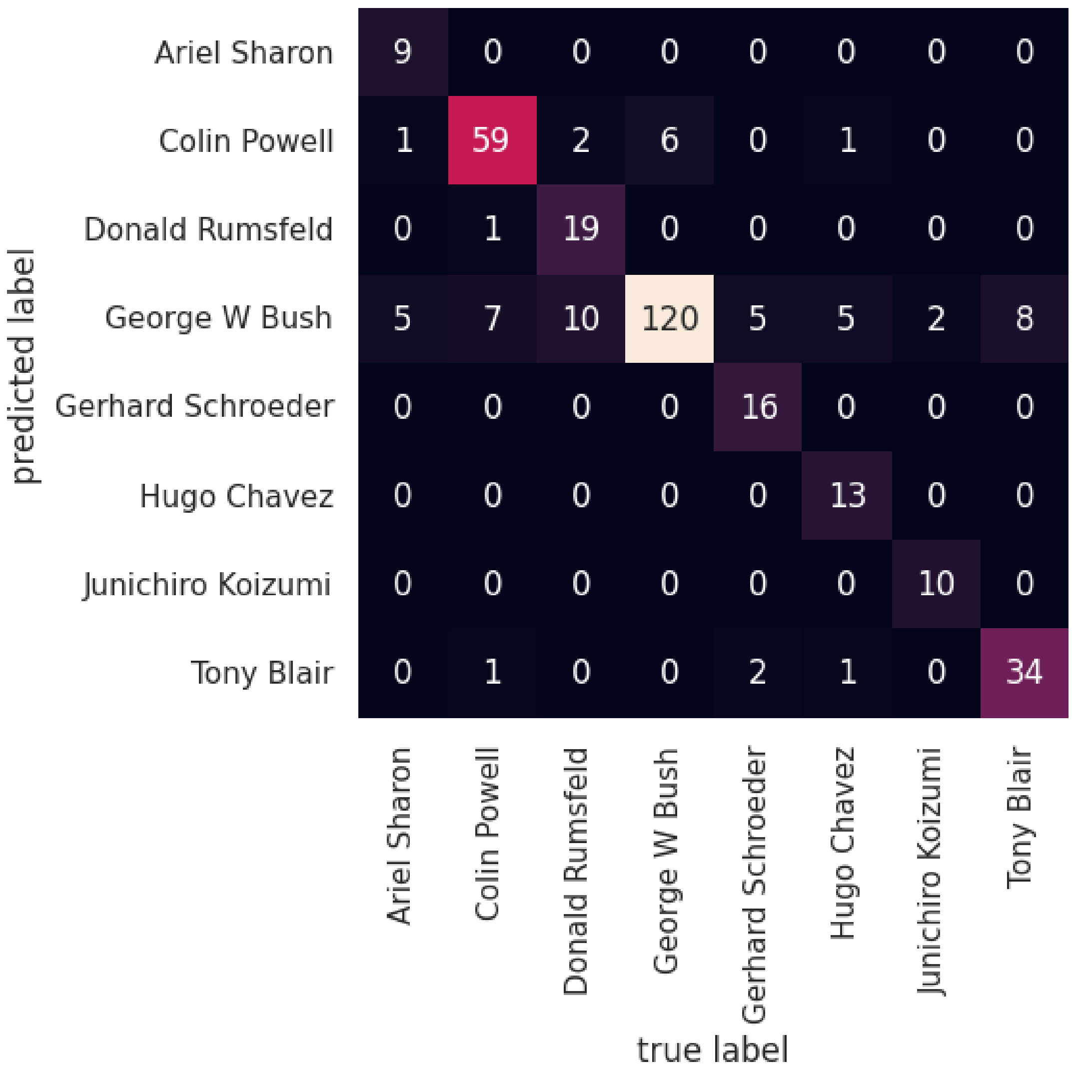
1.2 Some useful code for the midterm

[43]:
```
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_)

# 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': 5, 'svc__gamma': 0.001}
```






```
Accuracy =  0.8308605341246291
```