Sri Sivasubramaniya Nadar College of Engineering, Chennai
(An autonomous Institution affiliated to Anna University)

Degree & Branch	B.E. Computer Science & Engineering	Semester	VI
Subject Code & Name	UCS2612 – Machine Learning Algorithms Laboratory		
Academic Year	2025–2026 (Even)	Batch	2023–2027
Due Date	**27.01.2026**		

Experiment 2: Binary Classification using Naïve Bayes and K-Nearest Neighbors

Objective

To implement Naïve Bayes and K-Nearest Neighbors (KNN) classifiers for a binary classification problem, evaluate them using multiple performance metrics, visualize model behavior, and analyze overfitting, underfitting, and bias–variance characteristics.

Dataset

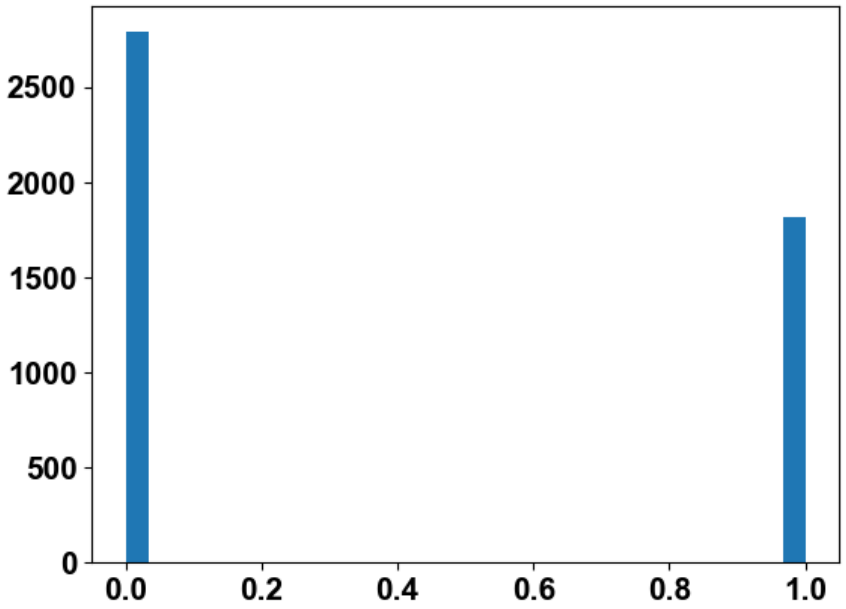
A benchmark binary classification dataset containing numerical features and two class labels is used.

Dataset reference:

- Kaggle: urlcolorSpambase Dataset

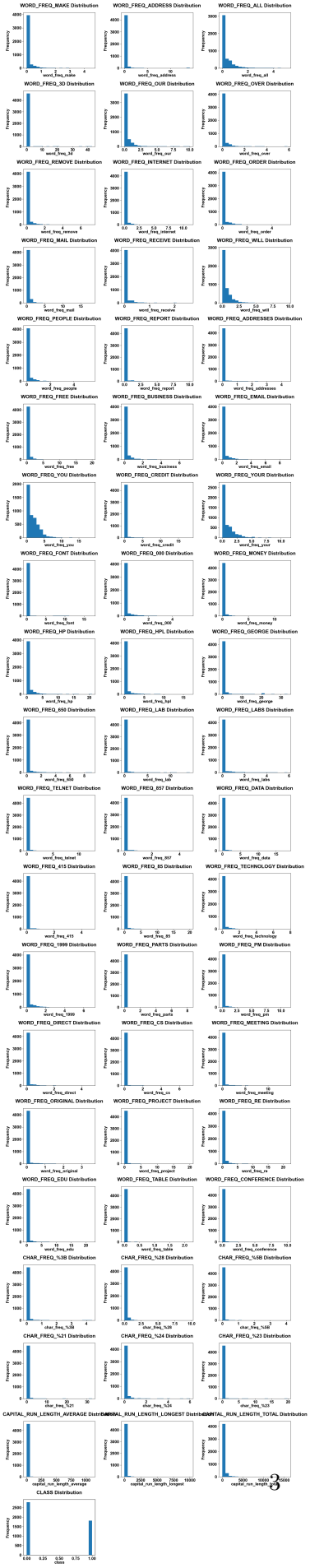
[23]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
import math
from matplotlib import rcParams
from sklearn.model_selection import train_test_split,␣
↪StratifiedKFold, GridSearchCV,␣
↪RandomizedSearchCV,cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB, MultinomialNB,␣
↪BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score,␣
↪recall_score, f1_score, confusion_matrix, roc_curve, auc
rcParams['font.family']='Arial'
rcParams['font.weight']='bold'
rcParams['font.size']=15
rcParams['axes.labelweight']='bold'
rcParams['axes.titleweight']='bold'
rcParams['xtick.labelsize']=15
```

```
plt.savefig("class_distribution.png", dpi=300,␣
↪bbox_inches="tight")
```



Histogram

[13]:
```
bxwidth=1
rows=math.ceil(len(df.columns)/3)
fig,axes=plt.subplots(rows,3,figsize=(15,4*rows))
axes=axes.flatten()
columns=df.columns
for i,(ax,col) in enumerate(zip(axes,columns)):
  ax.hist(df[col],bins=20)
  ax.set_title(f"{col.upper()} Distribution",pad=20)
  ax.set_xlabel(col,labelpad=0)
  ax.set_ylabel("Frequency",labelpad=10)
for spine in ax.spines.values():
  spine.set_linewidth(bxwidth)
for j in range(i+1,len(axes)):
  fig.delaxes(axes[j])
plt.subplots_adjust(hspace=1.5,wspace=1.3)
plt.tight_layout()
plt.savefig("histogram.png", dpi=300, bbox_inches="tight")
plt.show()
```


WORD_FREQ_MAKE Distribution
word_freq_make
WORD_FREQ_ADDRESS Distribution
word_freq_address
WORD_FREQ_ALL Distribution
word_freq_all
WORD_FREQ_3D Distribution
word_freq_3d
WORD_FREQ_OUR Distribution
word_freq_our
WORD_FREQ_OVER Distribution
word_freq_over
WORD_FREQ_REMOVE Distribution
word_freq_remove
WORD_FREQ_INTERNET Distribution
word_freq_internet
WORD_FREQ_ORDER Distribution
word_freq_order
WORD_FREQ_MAIL Distribution
word_freq_mail
WORD_FREQ_RECEIVE Distribution
word_freq_receive
WORD_FREQ_WILL Distribution
word_freq_will
WORD_FREQ_PEOPLE Distribution
word_freq_people
WORD_FREQ_REPORT Distribution
word_freq_report
WORD_FREQ_ADDRESSES Distribution
word_freq_addresses
WORD_FREQ_FREE Distribution
word_freq_free
WORD_FREQ_BUSINESS Distribution
word_freq_business
WORD_FREQ_EMAIL Distribution
word_freq_email
WORD_FREQ_YOU Distribution
word_freq_you
WORD_FREQ_CREDIT Distribution
word_freq_credit
WORD_FREQ_YOUR Distribution
word_freq_your
WORD_FREQ_FONT Distribution
word_freq_font
WORD_FREQ_000 Distribution
word_freq_000
WORD_FREQ_MONEY Distribution
word_freq_money
WORD_FREQ_HP Distribution
word_freq_hp
WORD_FREQ_HPL Distribution
word_freq_hpl
WORD_FREQ_GEORGE Distribution
word_freq_george
WORD_FREQ_650 Distribution
word_freq_650
WORD_FREQ_LAB Distribution
word_freq_lab
WORD_FREQ_LABS Distribution
word_freq_labs
WORD_FREQ_TELNET Distribution
word_freq_telnet
WORD_FREQ_857 Distribution
word_freq_857
WORD_FREQ_DATA Distribution
word_freq_data
WORD_FREQ_415 Distribution
word_freq_415
WORD_FREQ_85 Distribution
word_freq_85
WORD_FREQ_TECHNOLOGY Distribution
word_freq_technology
WORD_FREQ_1999 Distribution
word_freq_1999
WORD_FREQ_PARTS Distribution
word_freq_parts
WORD_FREQ_PM Distribution
word_freq_pm
WORD_FREQ_DIRECT Distribution
word_freq_direct
WORD_FREQ_CS Distribution
word_freq_cs
WORD_FREQ_MEETING Distribution
word_freq_meeting
WORD_FREQ_ORIGINAL Distribution
word_freq_original
WORD_FREQ_PROJECT Distribution
word_freq_project
WORD_FREQ_RE Distribution
word_freq_re
WORD_FREQ_EDU Distribution
word_freq_edu
WORD_FREQ_TABLE Distribution
word_freq_table
WORD_FREQ_CONFERENCE Distribution
word_freq_conference
CHAR_FREQ_%3B Distribution
char_freq_%3B
CHAR_FREQ_%28 Distribution
char_freq_%28
CHAR_FREQ_%5B Distribution
char_freq_%5B
CHAR_FREQ_%21 Distribution
char_freq_%21
CHAR_FREQ_%24 Distribution
char_freq_%24
CHAR_FREQ_%23 Distribution
char_freq_%23
CAPITAL_RUN_LENGTH_AVERAGE Distribution
capital_run_length_average
CAPITAL_RUN_LENGTH_LONGEST Distribution
capital_run_length_longest
CAPITAL_RUN_LENGTH_TOTAL Distribution
capital_run_length_total
CLASS Distribution
class
Frequency

Train-Test Split

[19]:
```
X=df.iloc[:,:-1]
y=df.iloc[:,-1]
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
↪2,stratify=y,random_state=42)
print(X_train,X_test,y_train,y_test)
```

Z-score or Standard Scaling

[21]:
```
scaler=StandardScaler()
X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.transform(X_test)
print(X_train,X_test,y_train,y_test)
```

Naive Bayes GaussianNB

[27]:
```
start=time.time()
gnb=GaussianNB()
gnb.fit(X_train_scaled,y_train)
ttgnb=time.time()-start
y_pred=gnb.predict(X_test_scaled)
print("The training time for Gaussian Naive Bayes is: ",ttgnb)
```

[39]:
```
start=time.time()
y_pred=gnb.predict(X_test_scaled)
ptgnb=time.time()-start
print("The Prediction time for Gaussian Naive Bayes is:␣
↪",ptgnb)
```

Accuracy

[30]:
```
accuracy_score(y_test,y_pred)
```

Precision

[31]:
```
precision_score(y_test,y_pred)
```

Recall

[32]:
```
recall_score(y_test,y_pred)
```

F1 Score

[33]:
```
f1_score(y_test,y_pred)
```

[34]:
```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

Specificity and False Positive Rate

[36]:
```
TN, FP, FN, TP = cm.ravel()
specificity = TN / (TN + FP)
false_positive_rate = FP / (FP + TN)

print("Specificity:", specificity)
print("False Positive Rate:", false_positive_rate)
```

[37]:
```
print("Overall Report of model␣
 ↪\n\n",classification_report(y_test,y_pred))
```

MultinomialNB

[40]:
```
start=time.time()
mnb=MultinomialNB()
mnb.fit(X_train,y_train)
ttmnb=time.time()-start


print("The training time for Multinomial Naive Bayes is:␣
 ↪",ttmnb)
```

```
The training time for Multinomial Naive Bayes is:  0.
 ↪009273767471313477
```

[41]:
```
start=time.time()
y_pred=mnb.predict(X_test)
ptmnb=time.time()-start
print("The Prediction time for Multinomial Naive Bayes is:␣
 ↪",ptmnb)
```

```
The Prediction time for Multinomial Naive Bayes is:  0.
 ↪009157180786132812
```

Accuracy

[42]:
```
accuracy_score(y_test,y_pred)
```

[42]:
```
0.7763300760043431
```

Precision Score

[43]:
```
precision_score(y_test,y_pred)
```

Recall-score

[44]:
```
recall_score(y_test,y_pred)
```

F1-score

[45]:
```
f1_score(y_test,y_pred)
```

[46]:
```
print("Confusion matrix is␣
↪\n",confusion_matrix(y_test,y_pred))
```

[48]:
```
TN, FP, FN, TP = cm.ravel()
specificity = TN / (TN + FP)
false_positive_rate = FP / (FP + TN)

print("Specificity:", specificity)
print("False Positive Rate:", false_positive_rate)
```

[47]:
```
print("Overall Report of model␣
↪\n\n",classification_report(y_test,y_pred))
```

BernoulliNB

[49]:
```
start=time.time()
bnb=BernoulliNB()
bnb.fit(X_train,y_train)
ttbnb=time.time()-start
print("The training time for Bernoulli Naive Bayes is:␣
↪",ttbnb)
```

[50]:
```
start=time.time()
y_pred=bnb.predict(X_test)
ptbnb=time.time()-start

print("The prediction time for Bernoulli Naive Bayes is:␣
↪",ptbnb)
```

```
The prediction time for Bernoulli Naive Bayes is:  0.
↪003880739212036133
```

Accuracy

[51]:
```
accuracy_score(y_test,y_pred)
```

Precision

[52]:
```
precision_score(y_test,y_pred)
```

Recall Score

[53]:
```
recall_score(y_test,y_pred)
```

F1-score

[54]:
```
f1_score(y_test,y_pred)
```

[56]:
```
print("Confusion matrix is␣
 ↪\n",confusion_matrix(y_test,y_pred))
```

```
Confusion matrix is
 [[515  43]
 [ 71 292]]
```

[57]:
```
TN, FP, FN, TP = cm.ravel()
specificity = TN / (TN + FP)
false_positive_rate = FP / (FP + TN)

print("Specificity:", specificity)
print("False Positive Rate:", false_positive_rate)
```

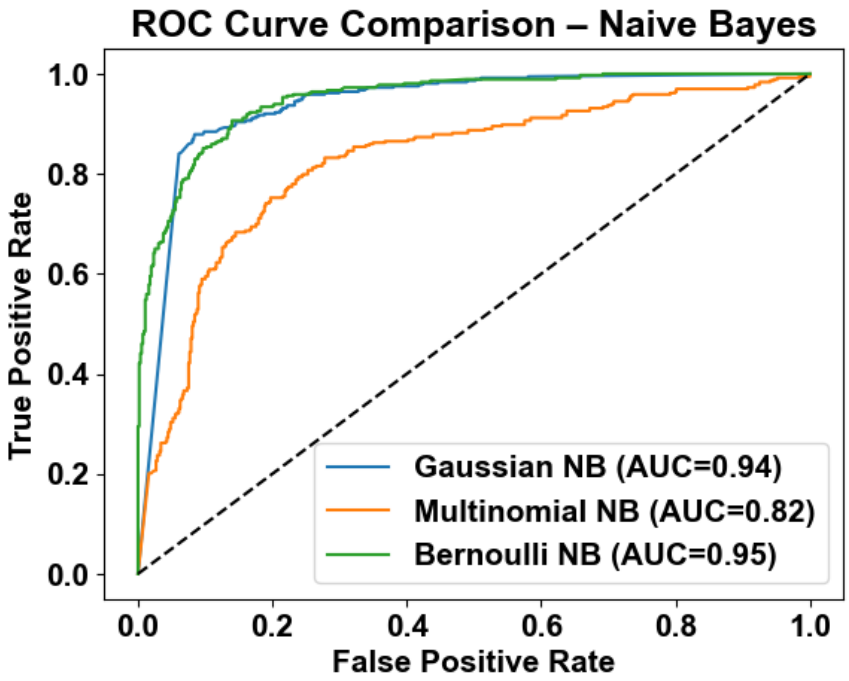
```
Specificity: 0.7508960573476703
False Positive Rate: 0.24910394265232974
```

[58]:
```
print("Overall Report of model␣
 ↪\n\n",classification_report(y_test,y_pred))
```

ROC curve

[59]:
```
y_prob_gnb=gnb.predict_proba(X_test_scaled)[:,1]
y_prob_mnb=mnb.predict_proba(X_test)[:,1]
y_prob_bnb=bnb.predict_proba(X_test)[:,1]
```

[60]:
```
fpr_gnb,tpr_gnb,_=roc_curve(y_test,y_prob_gnb)
fpr_mnb,tpr_mnb,_=roc_curve(y_test,y_prob_mnb)
fpr_bnb,tpr_bnb,_=roc_curve(y_test,y_prob_bnb)
auc_gnb=auc(fpr_gnb,tpr_gnb)
auc_mnb=auc(fpr_mnb,tpr_mnb)
auc_bnb=auc(fpr_bnb,tpr_bnb)
plt.plot(fpr_gnb,tpr_gnb,label=f'Gaussian NB (AUC={auc_gnb:.
 ↪2f})')
plt.plot(fpr_mnb,tpr_mnb,label=f'Multinomial NB (AUC={auc_mnb:
 ↪.2f})')
plt.plot(fpr_bnb,tpr_bnb,label=f'Bernoulli NB (AUC={auc_bnb:.
 ↪2f})')
plt.plot([0,1],[0,1],'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison - Naive Bayes")
plt.legend()
plt.savefig("roc_curve.png", dpi=300, bbox_inches="tight")
plt.show()
```



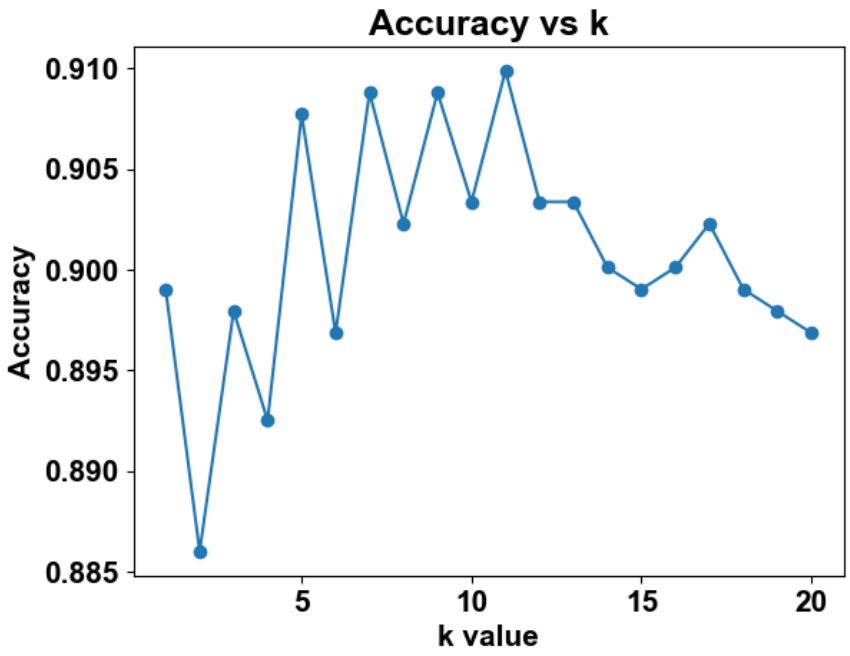
K-Nearest Neighbour

Basic Model

[61]:
```
knn = KNeighborsClassifier()
knn.fit(X_train_scaled, y_train)
y_pred_knn = knn.predict(X_test_scaled)
```

Statistical significance test Anova-N

[76]:
```
k_values = range(1, 21)
accuracies = []
for k in k_values:
  knn = KNeighborsClassifier(n_neighbors=k)
  knn.fit(X_train_scaled, y_train)
  accuracies.append(knn.score(X_test_scaled, y_test))
plt.plot(k_values, accuracies, marker='o')
plt.xlabel('k value')
plt.ylabel('Accuracy')
plt.title('Accuracy vs k')
plt.savefig("accuracy_vs_k.png", dpi=300, bbox_inches="tight")
plt.show()
```



Stratified K-Fold

[77]:
```
skf=StratifiedKFold(n_splits=5,shuffle=True,random_state=42)
cv_scores_base=cross_val_score(
    knn,
    X_train_scaled,
    y_train,
    cv=skf,
    scoring='accuracy'
)
print("Base KNN CV Accuracy:",cv_scores_base.mean())
```

Grid Search

[78]:
```
param_grid={
    'n_neighbors':list(range(1,31,2)),
    'weights':['uniform','distance'],
    'metric':['euclidean','manhattan']
}
grid=GridSearchCV(
    knn,
    param_grid,
    cv=skf,
    scoring='accuracy',
    n_jobs=-1
)
```

```
grid.fit(X_train_scaled,y_train)
print("Grid Best Params:",grid.best_params_)
print("Grid Best CV Accuracy:",grid.best_score_)
```

```
Grid Best Params: {'metric': 'manhattan', 'n_neighbors': 9,␣
↪'weights':
'distance'}
Grid Best CV Accuracy: 0.9252717391304348
```

Randomized Search

[79]:
```
from scipy.stats import randint
param_dist={
    'n_neighbors':randint(1,30),
    'weights':['uniform','distance'],
    'metric':['euclidean','manhattan']
}
rand=RandomizedSearchCV(
    knn,
    param_distributions=param_dist,
    n_iter=15,
    cv=skf,
    scoring='accuracy',
    random_state=42,
    n_jobs=-1
)
rand.fit(X_train_scaled,y_train)
print("Random Best Params:",rand.best_params_)
print("Random Best CV Accuracy:",rand.best_score_)
```

```
Random Best Params: {'metric': 'manhattan', 'n_neighbors': 6,␣
↪'weights':
'distance'}
Random Best CV Accuracy: 0.9241847826086957
```

Final KNN Model

[80]:
```
best_params=grid.best_params_
knn_final=KNeighborsClassifier(
    n_neighbors=best_params['n_neighbors'],
    weights=best_params['weights'],
    metric=best_params['metric']
)
knn_final.fit(X_train_scaled,y_train)
y_pred_final=knn_final.predict(X_test_scaled)
```

Metrics

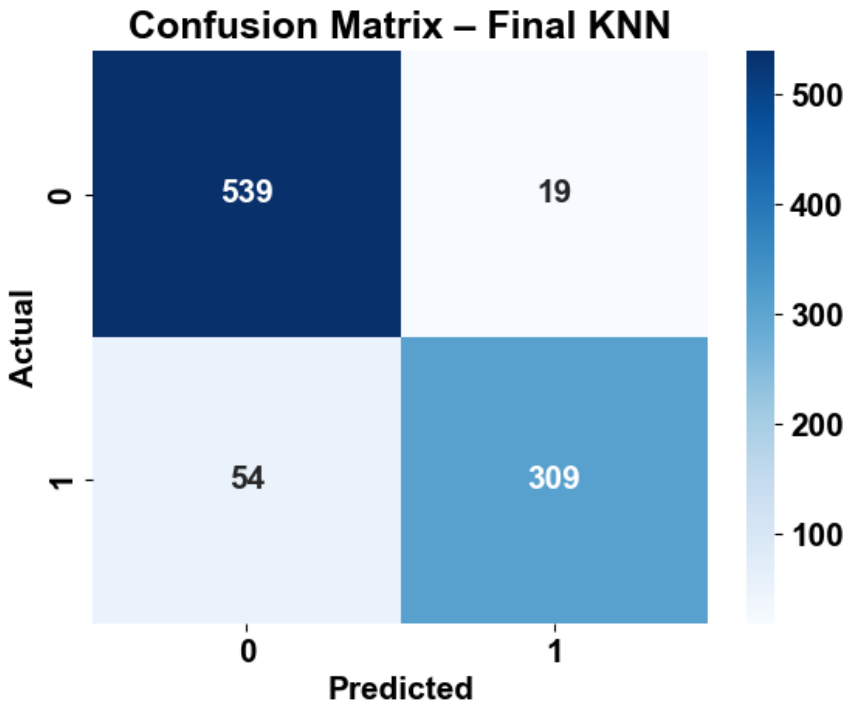
[81]:
```
from sklearn.metrics import confusion_matrix,roc_curve,auc
def compute_metrics(y_true,y_pred):
    cm=confusion_matrix(y_true,y_pred)
    tn,fp,fn,tp=cm.ravel()
    accuracy=(tp+tn)/(tp+tn+fp+fn)
    precision=tp/(tp+fp)
    recall=tp/(tp+fn)
    f1=2*precision*recall/(precision+recall)
    specificity=tn/(tn+fp)
    fpr=fp/(fp+tn)
    return accuracy,precision,recall,f1,specificity,fpr,cm
```

[82]:
```
start=time.time()
knn_final.fit(X_train_scaled,y_train)
train_time=time.time()-start

start=time.time()
y_pred_knn=knn_final.predict(X_test_scaled)
pred_time=time.time()-start
acc,prec,rec,f1,spec,fpr,cm=compute_metrics(y_test,y_pred_knn)
print("Final KNN Metrics")
print("Accuracy:",acc)
print("Precision:",prec)
print("Recall:",rec)
print("F1 Score:",f1)
print("Specificity:",spec)
print("False Positive Rate:",fpr)
print("Training Time:",train_time)
print("Prediction Time:",pred_time)
```

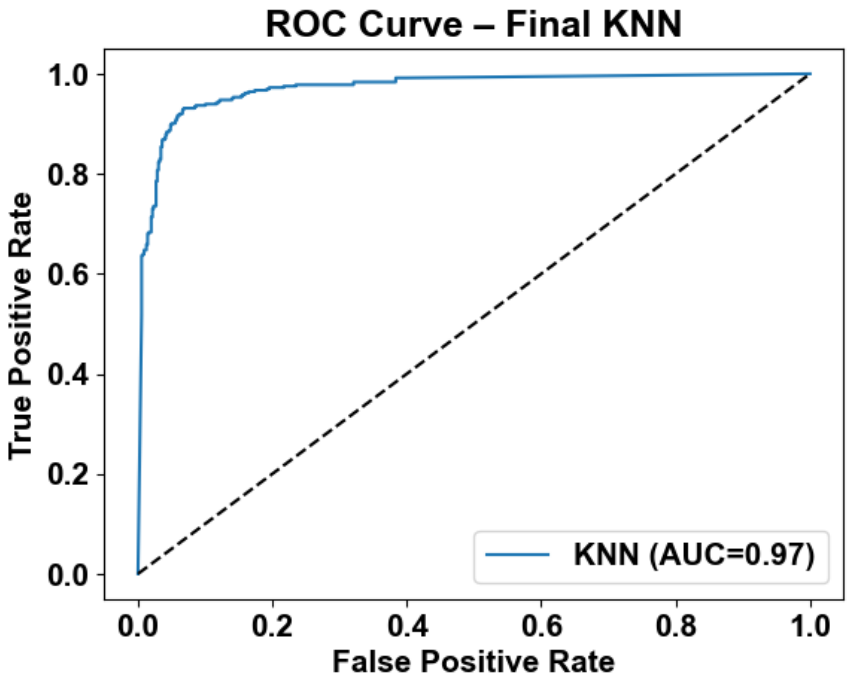
Confustion Matrix

[83]:
```
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.title("Confusion Matrix - Final KNN")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



ROC Curve for KNN

[85]:
```
y_prob_knn=knn_final.predict_proba(X_test_scaled)[:,1]
fpr_knn,tpr_knn,_=roc_curve(y_test,y_prob_knn)
auc_knn=auc(fpr_knn,tpr_knn)
plt.plot(fpr_knn,tpr_knn,label=f'KNN (AUC={auc_knn:.2f})')
plt.plot([0,1],[0,1],'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Final KNN")
plt.legend()
plt.savefig("roc_curve_finalKNN.png", dpi=300,␣
↪bbox_inches="tight")
plt.show()
```



[86]:
```
best_params=grid.best_params_
optimal_k=best_params['n_neighbors']
```

Evaluating all models using multiple metrics

KDTree

[87]:
```
start=time.time()
knn_kd=KNeighborsClassifier(
    n_neighbors=optimal_k,
    weights=best_params['weights'],
    metric=best_params['metric'],
    algorithm='kd_tree'
)
knn_kd.fit(X_train_scaled,y_train)
train_time_kd=time.time()-start
start=time.time()
y_pred_kd=knn_kd.predict(X_test_scaled)
pred_time_kd=time.time()-start
```

[88]:
```
acc_kd,prec_kd,rec_kd,f1_kd,sp,fpr,cm=compute_metrics(y_test,y_pred_kd)
print("Final KDtree Metrics")
print("Accuracy:",acc_kd)
print("Precision:",prec_kd)
print("Recall:",rec_kd)
```

```
print("F1 Score:",f1_kd)
print("Specificity:",sp)
print("False Positive Rate:",fpr)
print("Training Time:",train_time)
print("Prediction Time:",pred_time)
```

BallTree

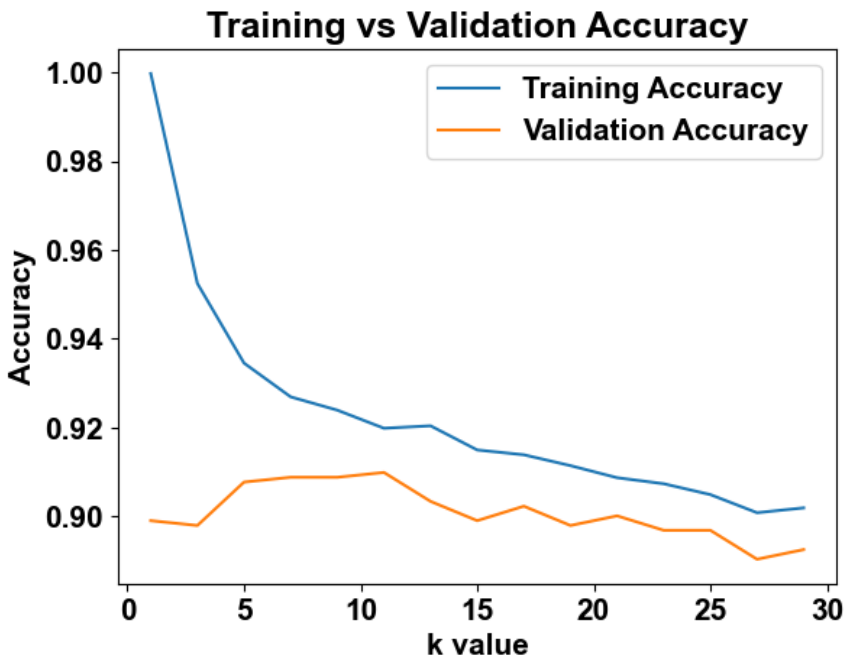
[89]:
```
start=time.time()
knn_bt=KNeighborsClassifier(
    n_neighbors=optimal_k,
    weights=best_params['weights'],
    metric=best_params['metric'],
    algorithm='ball_tree'
)
knn_bt.fit(X_train_scaled,y_train)
train_time_bt=time.time()-start
start=time.time()
y_pred_bt=knn_bt.predict(X_test_scaled)
pred_time_bt=time.time()-start
```

[90]:
```
acc_bt,prec_bt,rec_bt,f1_bt,sp,fpr,cm=compute_metrics(y_test,y_pred_bt)
print("Final Balltree Metrics")
print("Accuracy:",acc_bt)
print("Precision:",prec_bt)
print("Recall:",rec_bt)
print("F1 Score:",f1_bt)
print("Specificity:",sp)
print("False Positive Rate:",fpr)
print("Training Time:",train_time)
print("Prediction Time:",pred_time)
```

Training vs Validation

[92]:
```
train_acc=[]
val_acc=[]
k_values=range(1,31,2)
for k in k_values:
    knn=KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled,y_train)
    train_acc.append(knn.score(X_train_scaled,y_train))
    val_acc.append(knn.score(X_test_scaled,y_test))
plt.plot(k_values,train_acc,label='Training Accuracy')
plt.plot(k_values,val_acc,label='Validation Accuracy')
plt.xlabel('k value')
plt.ylabel('Accuracy')
plt.title('Training vs Validation Accuracy')
plt.legend()
```

```
plt.savefig("training_vs_validation_accuracy.png", dpi=300,␣
↪bbox_inches="tight")
plt.show()
```



Cross Validation Scores

[102]:
```
gnb_scores=cross_val_score(
    GaussianNB(),
    X_train,
    y_train,
    cv=skf,
    scoring='precision'
)
```

[103]:
```
mnb_scores=cross_val_score(
    MultinomialNB(),
    X_train,
    y_train,
    cv=skf,
    scoring='precision'
)
```

[104]:
```
bnb_scores=cross_val_score(
    BernoulliNB(),
    X_train,
```

```
    y_train,
    cv=skf,
    scoring='precision'
)
```

[105]:
```
knn_kd_scores=cross_val_score(
    knn_kd,
    X_train_scaled,
    y_train,
    cv=skf,
    scoring='precision'
)
```

[106]:
```
knn_bt_scores=cross_val_score(
    knn_bt,
    X_train_scaled,
    y_train,
    cv=skf,
    scoring='precision'
)
```

One way ANOVA Test

[107]:
```
from scipy.stats import f_oneway
F_stat,p_value=f_oneway(
    gnb_scores,
    mnb_scores,
    bnb_scores,
    knn_kd_scores,
    knn_bt_scores
)

print("F-statistic:",F_stat)
print("p-value:",p_value)
```

```
F-statistic: 147.4451411146714
p-value: 1.537483731179684e-14
```

Mean Accuracy to find the best model

[108]:
```
print("Gaussian NB Mean precision:",gnb_scores.mean())
print("Multinomial NB Mean precision:",mnb_scores.mean())
print("Bernoulli NB Mean precision:",bnb_scores.mean())
print("KNN KDTree Mean precision:",knn_kd_scores.mean())
print("KNN BallTree Mean precision:",knn_bt_scores.mean())
```

[109]:
```
best_model=max(
    [
        ("Gaussian NB",gnb_scores.mean()),
```

```
        ("Multinomial NB",mnb_scores.mean()),
        ("Bernoulli NB",bnb_scores.mean()),
        ("KNN KDTree",knn_kd_scores.mean()),
        ("KNN BallTree",knn_bt_scores.mean())
    ],
    key=lambda x:x[1]
)

print("Best Model:",best_model)
```

```
Best Model: ('KNN KDTree', np.float64(0.948881300360718))
```

[ ]:

Naïve Bayes Performance Comparison

Metric	Gaussian NB	Multinomial NB	Bernoulli NB
Accuracy	0.8327	0.7763	0.8762
Precision	0.7145	0.7198	0.8716
Recall	0.9586	0.7079	0.8044
F1 Score	0.8188	0.7138	0.8366
Specificity	0.7508	0.7508	0.7508
Training Time (s)	0.0046	0.0092	0.0256

KNN Hyperparameter Tuning Results

Search Method	Best k	Best CV Accuracy	Best Parameters
Grid Search	9	0.9252	'metric': 'manhattan', 'weights': 'distance'
Randomized Search	6	0.9241	'metric': 'manhattan', 'weights': 'distance'

KNN Performance using Different Search Methods

KDTree vs BallTree Comparison

Conclusion

Naïve Bayes provides fast and stable performance with high bias, whereas optimized KNN achieves better accuracy and generalization through careful

Metric (KDTree)	Value
Optimal k	9
Accuracy	0.9207
Precision	0.9420
Recall	0.8512
F1 Score	0.8943
Training Time (s)	0.0056
Prediction Time (s)	0.0406

Metric (BallTree)	Value
Optimal k	9
Accuracy	0.9207
Precision	0.9420
Recall	0.8512
F1 Score	0.8943
Training Time (s)	0.0056
Prediction Time (s)	0.0406

Criterion	KDTree	BallTree
Accuracy	0.9207	0.9207
Training Time (s)	0.0056	0.0056
Prediction Time (s)	0.0406	0.0406
Memory Usage	Low / Medium	Medium / High

hyperparameter tuning, with KDTree and BallTree improving computational efficiency.

References

- urlcolorScikit-learn: Naïve Bayes
- urlcolorScikit-learn: KNN
- urlcolorScikit-learn: Hyperparameter Optimization
- urlcolorSpambase Dataset