FP

April 23, 2024

0.1 Import modules

[1]:

```
# Import standard DS packages
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import math
import scipy
import statistics
import textwrap
%matplotlib inline
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('svg')


from sklearn.decomposition import PCA
from sklearn.feature_selection import VarianceThreshold,SelectKBest,chi2
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn import metrics
from sklearn.feature_selection import SelectPercentile
from sklearn import neighbors
from sklearn.model_selection import GridSearchCV
from sklearn import tree
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import precision_score, recall_score,␣
  ↪f1_score,confusion_matrix, ConfusionMatrixDisplay

from warnings import simplefilter
simplefilter(action='ignore')
```

```
/tmp/ipykernel_1824278/195150086.py:13: DeprecationWarning:
`set_matplotlib_formats` is deprecated since IPython 7.23, directly use
`matplotlib_inline.backend_inline.set_matplotlib_formats()`
```

```
 set_matplotlib_formats('svg')
```

0.2 Load the data

```
[2]: permissions = pd.read_csv('data/permissions.csv')
     permissions.head()
```

```
[2]:    android.permission.GET_ACCOUNTS  \
     0                                0   
     1                                0   
     2                                0   
     3                                0   
     4                                0   

        com.sonyericsson.home.permission.BROADCAST_BADGE  \
     0                                                 0   
     1                                                 0   
     2                                                 0   
     3                                                 0   
     4                                                 0   

        android.permission.READ_PROFILE  android.permission.MANAGE_ACCOUNTS  \
     0                                0                                   0   
     1                                0                                   0   
     2                                0                                   0   
     3                                0                                   0   
     4                                0                                   0   

        android.permission.WRITE_SYNC_SETTINGS  \
     0                                       0   
     1                                       0   
     2                                       0   
     3                                       0   
     4                                       0   

        android.permission.READ_EXTERNAL_STORAGE  android.permission.RECEIVE_SMS  \
     0                                         0                               0   
     1                                         1                               0   
     2                                         0                               0   
     3                                         0                               0   
     4                                         0                               0   

        com.android.launcher.permission.READ_SETTINGS  \
     0                                              0   
     1                                              0   
     2                                              0   
```

```
3                                              0
4                                              0

   android.permission.WRITE_SETTINGS  \
0                                  0
1                                  0
2                                  0
3                                  0
4                                  0

   com.google.android.providers.gsf.permission.READ_GSERVICES  …  \
0                                                  0           …
1                                                  0           …
2                                                  0           …
3                                                  0           …
4                                                  0           …

   com.android.launcher.permission.UNINSTALL_SHORTCUT  \
0                                                  0
1                                                  0
2                                                  0
3                                                  0
4                                                  0

   com.sec.android.iap.permission.BILLING  \
0                                       0
1                                       0
2                                       0
3                                       0
4                                       0

   com.htc.launcher.permission.UPDATE_SHORTCUT  \
0                                            0
1                                            0
2                                            0
3                                            0
4                                            0

   com.sec.android.provider.badge.permission.WRITE  \
0                                                0
1                                                0
2                                                0
3                                                0
4                                                0

   android.permission.ACCESS_NETWORK_STATE  \
0                                        0
```

```
1                                         1
2                                         1
3                                         1
4                                         1

   com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE  \
0                                                   0
1                                                   0
2                                                   0
3                                                   0
4                                                   1

   com.huawei.android.launcher.permission.READ_SETTINGS   \
0                                                   0
1                                                   0
2                                                   0
3                                                   0
4                                                   0

   android.permission.READ_SMS  android.permission.PROCESS_INCOMING_CALLS  \
0                            0                                          0
1                            0                                          0
2                            0                                          0
3                            0                                          0
4                            0                                          0

   Result
0       0
1       0
2       0
3       0
4       0

[5 rows x 87 columns]
```

[3]:

```
#Check for missing values in the data
print("Missing values in the data: ",sum(list(permissions.isna().sum())))
```

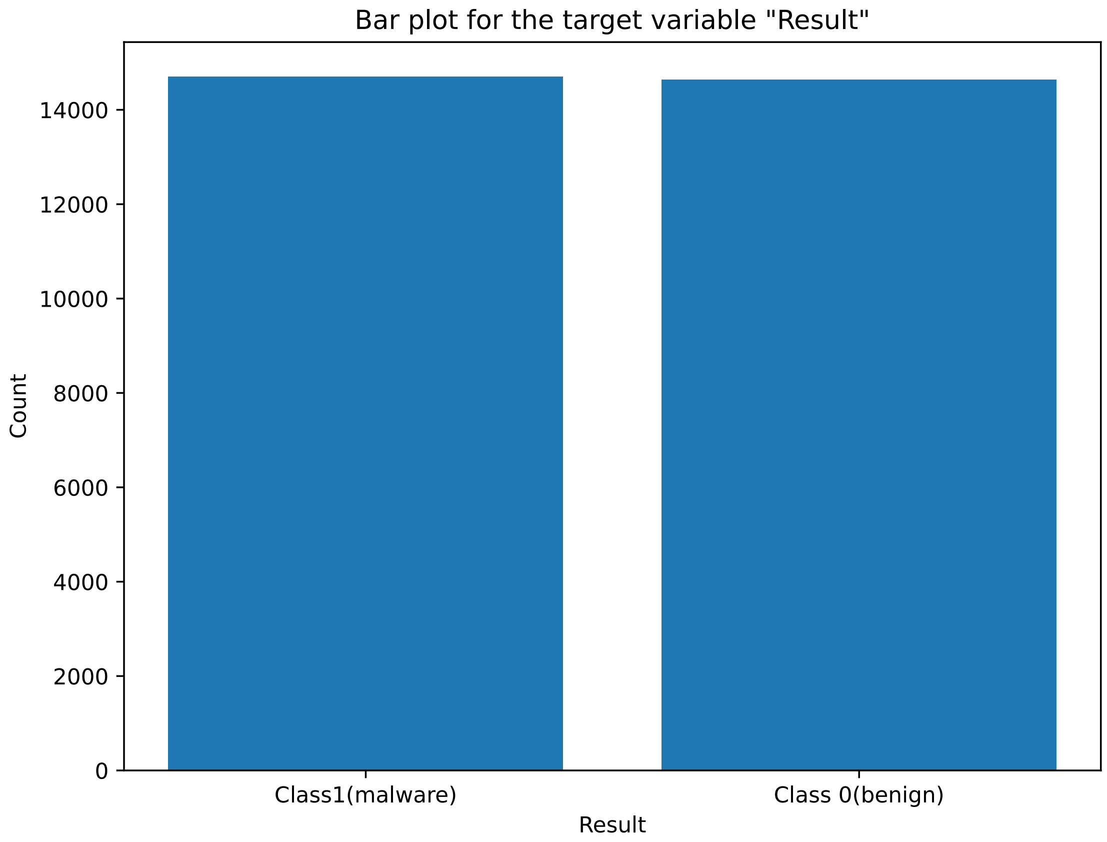
```
Missing values in the data:  0
```

[4]:

```
## Separating target variable form the data
X=permissions.iloc[:,:-1]
y=permissions.iloc[:,86]
```

[5]:

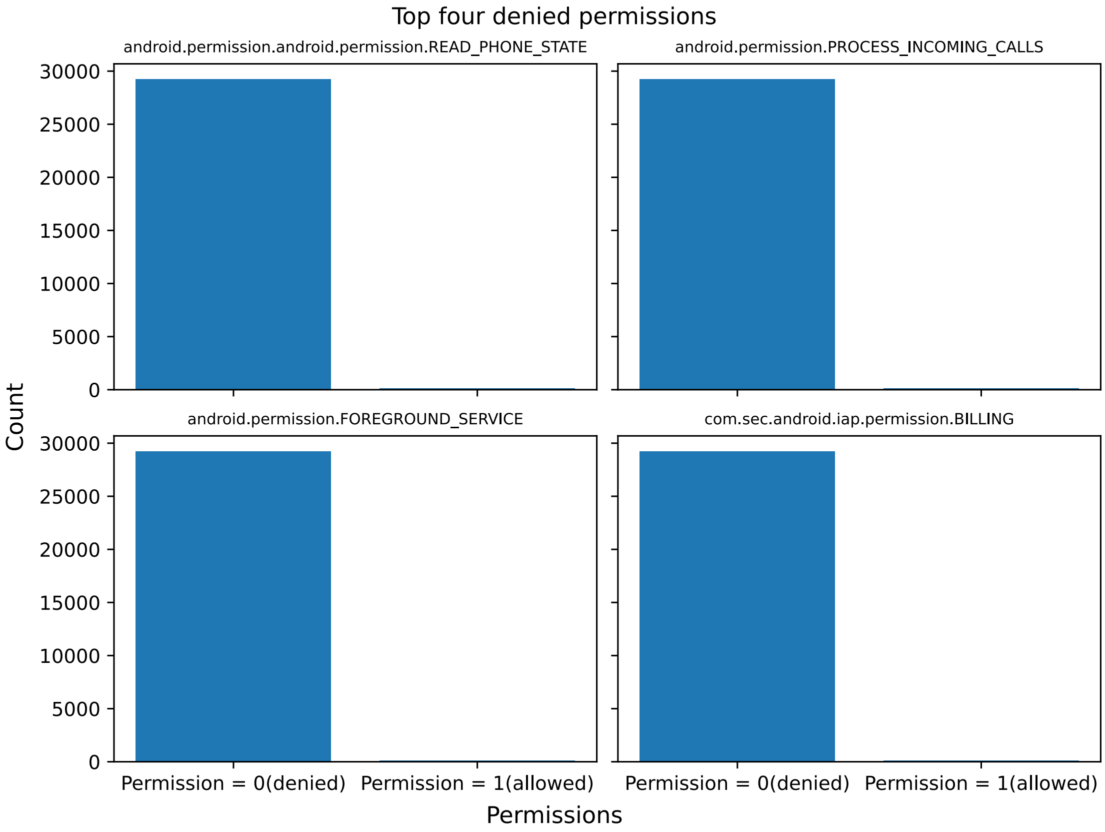
```
# Create bar plot for "Result" to show distribution of "Result"
plt.figure(figsize=(8,6))
plt.bar(["Class "+str(i)+"(benign)" if i==0 else "Class"+str(i)+"(malware)" for␣
 ↪i in y.value_counts().index ],y.value_counts())
plt.xlabel("Result")
plt.ylabel("Count")

plt.title("Bar plot for the target variable \"Result\"")
plt.rcParams['figure.dpi'] = 300
plt.rcParams['savefig.dpi'] = 300
plt.show()
```



[6]:
```
#Class percentage
print("Percentage of data with clas=0(benign), ",round((y.value_counts()[0]/(y.
 ↪value_counts()[1]+y.value_counts()[0]))*100,2),"%.")
print("Percentage of data with class=1(malware), ",round((y.value_counts()[1]/
 ↪(y.value_counts()[0]+y.value_counts()[1]))*100,2),"%")
```

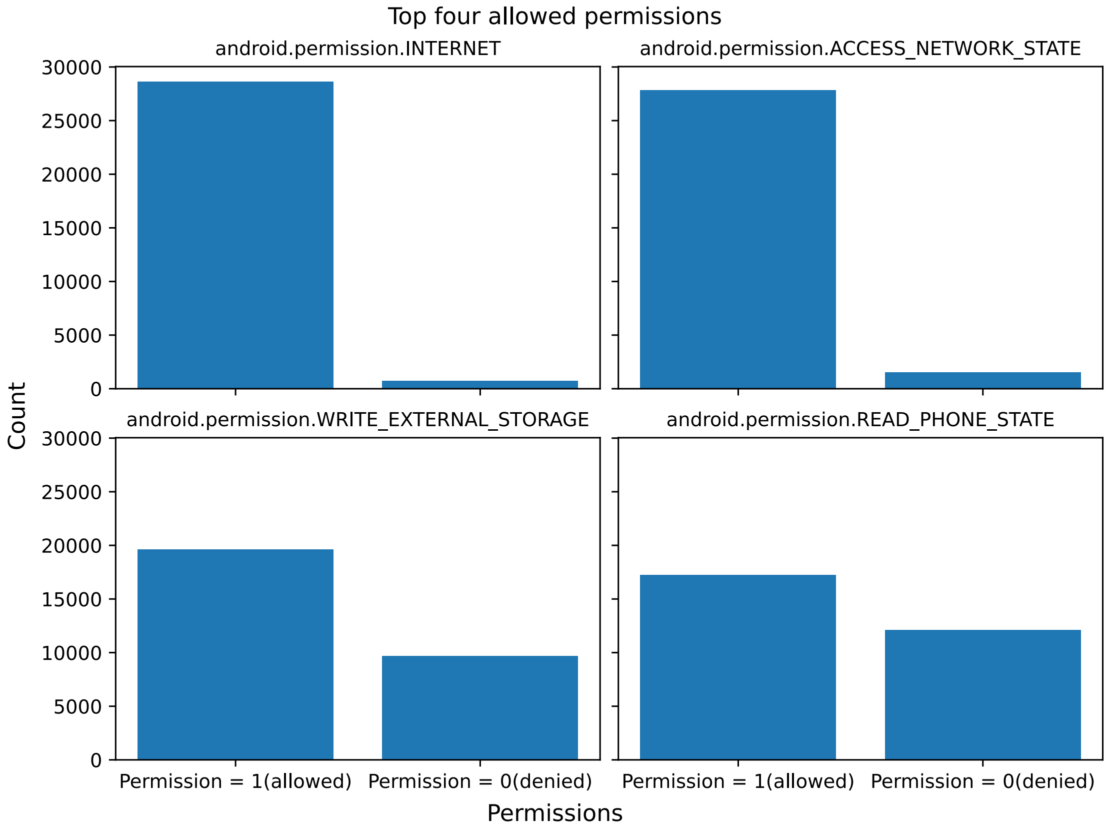
```
Percentage of data with clas=0(benign),  49.88 %.
Percentage of data with class=1(malware),  50.12 %
```

[7]:
```python
#Barplot for top four permisssions denied
figure, axis = plt.subplots(2, 2,figsize=(8, 6),sharex=True, sharey=True,␣
 ↪layout="constrained")
count=0
for i in range(2):
    for j in range(2):
        count+=1
        axis[i,j].bar(["Permission = "+str(i)+"(denied)" if i==0 else␣
 ↪"Permission = "+str(i)+"(allowed)" for i in permissions[X.sum().
 ↪sort_values(ascending=False)[-4:].index[-count]].value_counts().
 ↪index],permissions[X.sum().sort_values(ascending=False)[-4:].index[-count]].
 ↪value_counts())
        axis[i,j].set_title(X.sum().sort_values(ascending=False)[-4:].
 ↪index[-count],fontsize=8)
figure.supxlabel("Permissions")
figure.supylabel("Count")
figure.suptitle("Top four denied permissions")
plt.show()
```

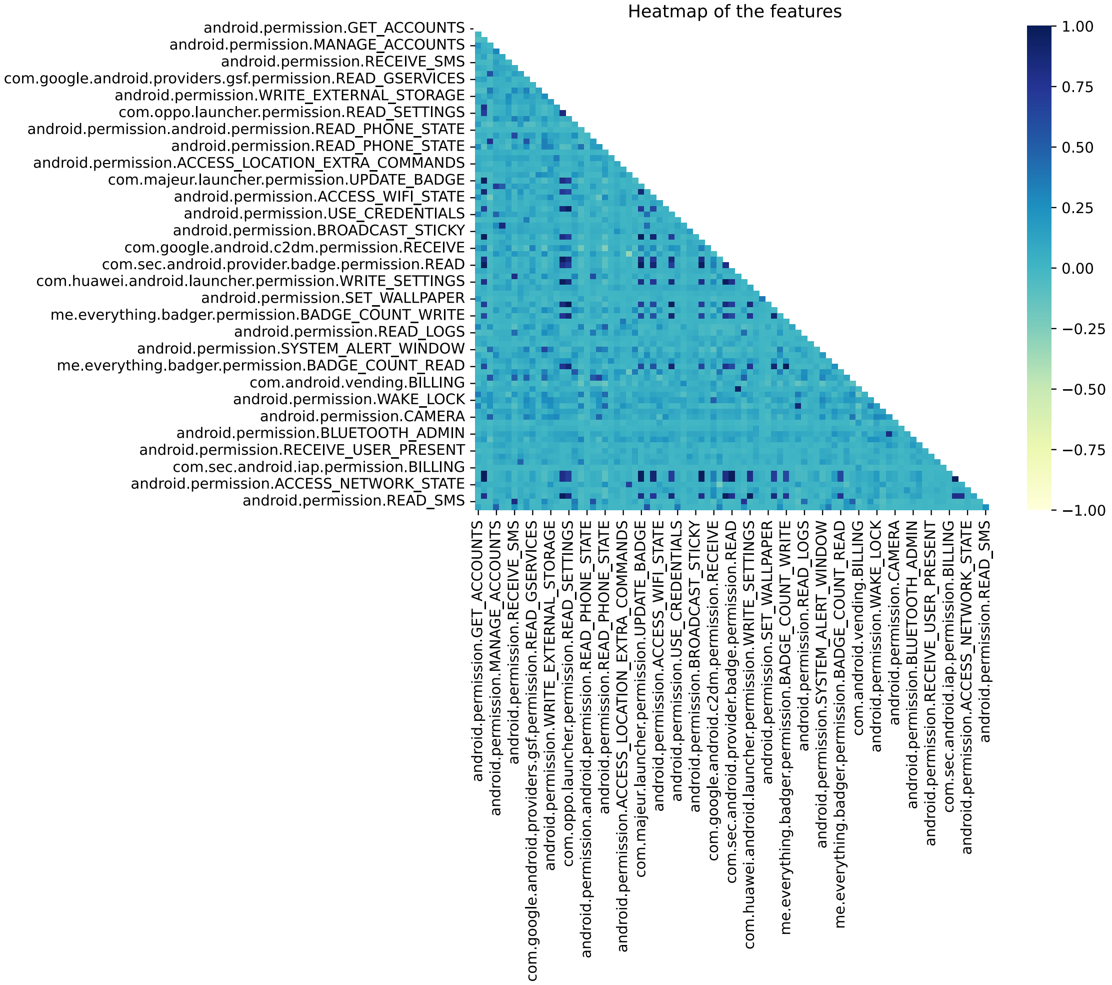


[9]:

```
#Barplot for top four permisssions allowed
figure, axis = plt.subplots(2, 2,figsize=(8, 6),sharex=True, sharey=True,␣
 ↪layout="constrained")
count=0
for i in range(2):
    for j in range(2):
        axis[i,j].bar(["Permission = "+str(i)+"(denied)" if i==0 else␣
 ↪"Permission = "+str(i)+"(allowed)" for i in permissions[X.sum().
 ↪sort_values(ascending=False)[:4].index[count]].value_counts().
 ↪index],permissions[X.sum().sort_values(ascending=False)[:4].index[count]].
 ↪value_counts())
        axis[i,j].set_title(X.sum().sort_values(ascending=False)[:4].
 ↪index[count],fontsize=10)
        count+=1
figure.supxlabel("Permissions")
figure.supylabel("Count")
figure.suptitle("Top four allowed permissions")
plt.show()
```



```
[12]: #Correlation heatmap
      plt.figure(figsize=(8,6))
      cm = X.corr()
      mask = np.triu(np.ones_like(cm, dtype=bool))
      sns.heatmap(data=cm,vmin=-1, vmax=1,mask=mask,cmap='YlGnBu',annot=True,fmt=".
       ↪2f")
      plt.title("Heatmap of the features",fontsize=12)
      plt.show()
```



0.3 Variance Threshold

```
[13]: #Variance Threshold with threshold=0.05
      selector = VarianceThreshold(threshold=0.05).fit(X)
      var_X=selector.transform(X)
```

[14]:

```
#Selecting appropriate features
cor_X=X.iloc[:,selector.get_support(indices=True)]
```

[15]:

```
# Split of the train and test set
X_trainval_var, X_test_var, y_trainval_var, y_test_var =␣
 ↪train_test_split(cor_X, y, test_size=0.2, stratify=y, random_state=100)

# Split trainval into train + val
X_train_var, X_val_var, y_train_var, y_val_var =␣
 ↪train_test_split(X_trainval_var, y_trainval_var, test_size=0.25,␣
 ↪stratify=y_trainval_var, random_state=100)
```

[16]:

```
#Setting scoring metrics
scoring = {
    'AUC': 'roc_auc',
    'Accuracy': metrics.make_scorer(metrics.accuracy_score)
}
```

[17]:

```
#Random forest hyperparameter tuning with variance threshold
rfVar_pipe = Pipeline([('rf1','passthrough')])
rfVar_params =[
               {'rf1':
 ↪[RandomForestClassifier(random_state=100)],'rf1__n_estimators':range(50),
                'rf1__max_features':[5,10,15,20,25]}

              ]

cvStrat = StratifiedKFold(n_splits=10, shuffle=True, random_state=100)

rfVar_grid = GridSearchCV(rfVar_pipe, rfVar_params, cv=cvStrat,␣
 ↪scoring=scoring,refit="Accuracy").fit(X_trainval_var,y_trainval_var)


# Report the best hyper-parameters and final test set performance
rfVar_best_params = rfVar_grid.best_params_

train_acc_rfVar = metrics.accuracy_score(y_trainval_var,rfVar_grid.
 ↪best_estimator_.predict(X_trainval_var))
test_acc_rfVar = metrics.accuracy_score(y_test_var,rfVar_grid.best_estimator_.
 ↪predict(X_test_var))
```

[18]:

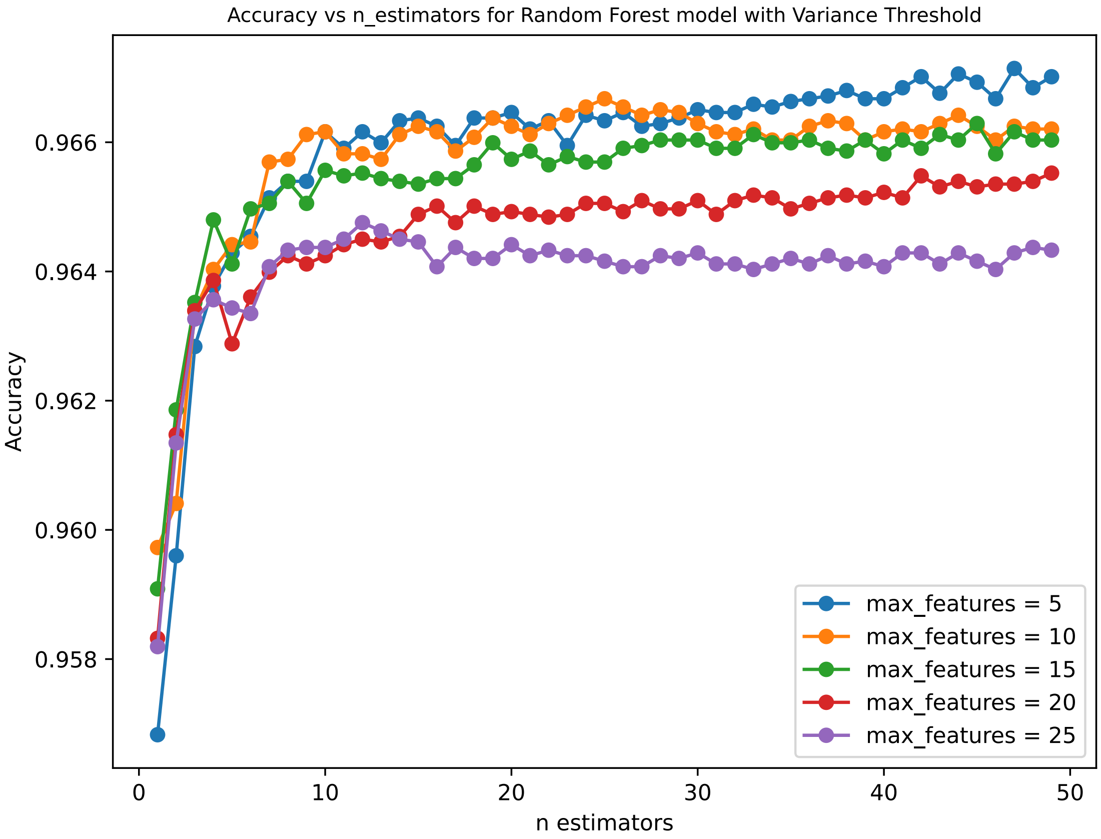
```
print("Best paramters for Random Forest model with Variance␣
 ↪Threshold",rfVar_best_params,"\n")
print("Accuracy for Random Forest model with Variance␣
 ↪Threshold",test_acc_rfVar,train_acc_rfVar)
```

```
Best paramters for Random Forest model with Variance Threshold {'rf1':
```

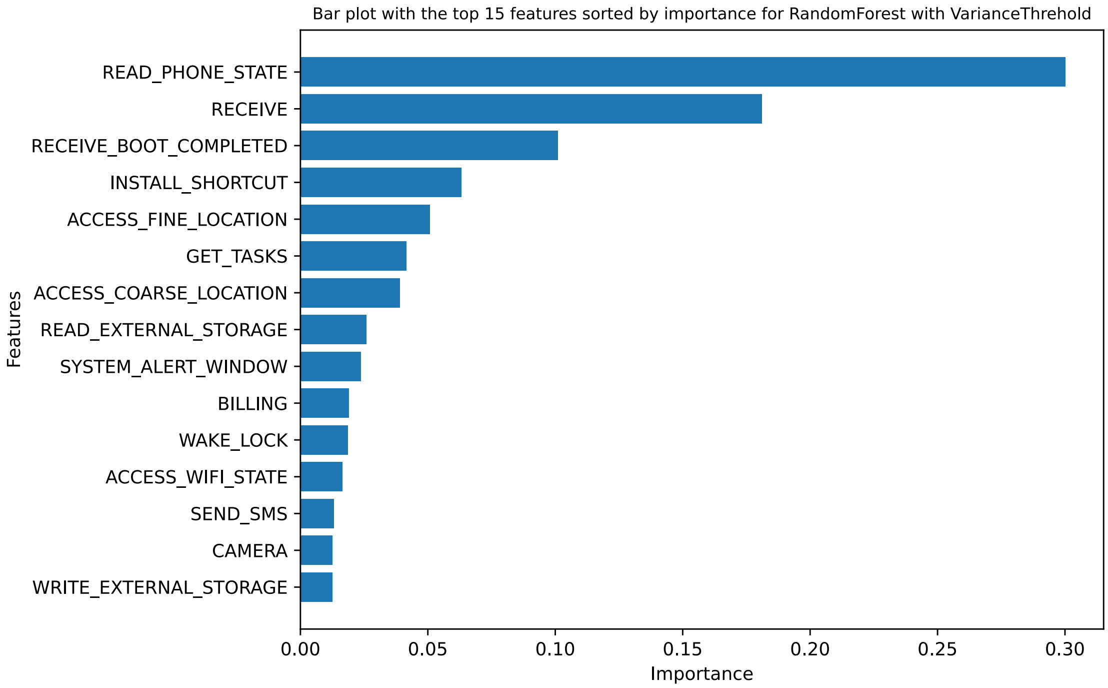
```
RandomForestClassifier(max_features=5, n_estimators=47, random_state=100),
'rf1__max_features': 5, 'rf1__n_estimators': 47}

Accuracy for Random Forest model with Variance Threshold 0.9618203511164138
0.9787342851054762
```

[19]:

```
#Tunining plot for Random Forest with varince threshold
n_estimators_list= range(50)
plt.figure(figsize=(8, 6))

plt.plot(n_estimators_list, rfVar_grid.cv_results_['mean_test_Accuracy'][:
 ↪50],marker="o", label='Train Accuracy')
plt.plot(n_estimators_list, rfVar_grid.cv_results_['mean_test_Accuracy'][50:
 ↪100],marker="o", label='Train Accuracy')
plt.plot(n_estimators_list, rfVar_grid.cv_results_['mean_test_Accuracy'][100:
 ↪150],marker="o", label='Train Accuracy')
plt.plot(n_estimators_list, rfVar_grid.cv_results_['mean_test_Accuracy'][150:
 ↪200],marker="o", label='Train Accuracy')
plt.plot(n_estimators_list, rfVar_grid.cv_results_['mean_test_Accuracy'][200:
 ↪250],marker="o", label='Train Accuracy')
plt.legend(['max_features = 5','max_features = 10','max_features =␣
 ↪15','max_features = 20','max_features = 25'])

plt.xlabel('n estimators')
plt.ylabel('Accuracy')
plt.title('Accuracy vs n_estimators for Random Forest model with Variance␣
 ↪Threshold',fontsize=9)
plt.show()
```



```
[20]: # Calculate performance on test data `auc_test`
rfVar_auc_test = metrics.roc_auc_score(y_test_var,rfVar_grid.best_estimator_.
 ↪predict(X_test_var))

importance = rfVar_grid.best_estimator_.named_steps.rf1.feature_importances_
imp = pd.DataFrame({'feature':cor_X.columns,'importance':importance}).
 ↪sort_values(by="importance", ascending=False)
# Create plot of top 15 features sorted by importance.
plt.figure(figsize=(8, 6))

plt.barh([i.split('.')[-1] for i in imp[:15].feature[::-1].tolist()],imp[:15].
 ↪importance[::-1])
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title('Bar plot with the top 15 features sorted by importance for␣
 ↪RandomForest with VarianceThrehold',fontsize=9)
plt.show()
```



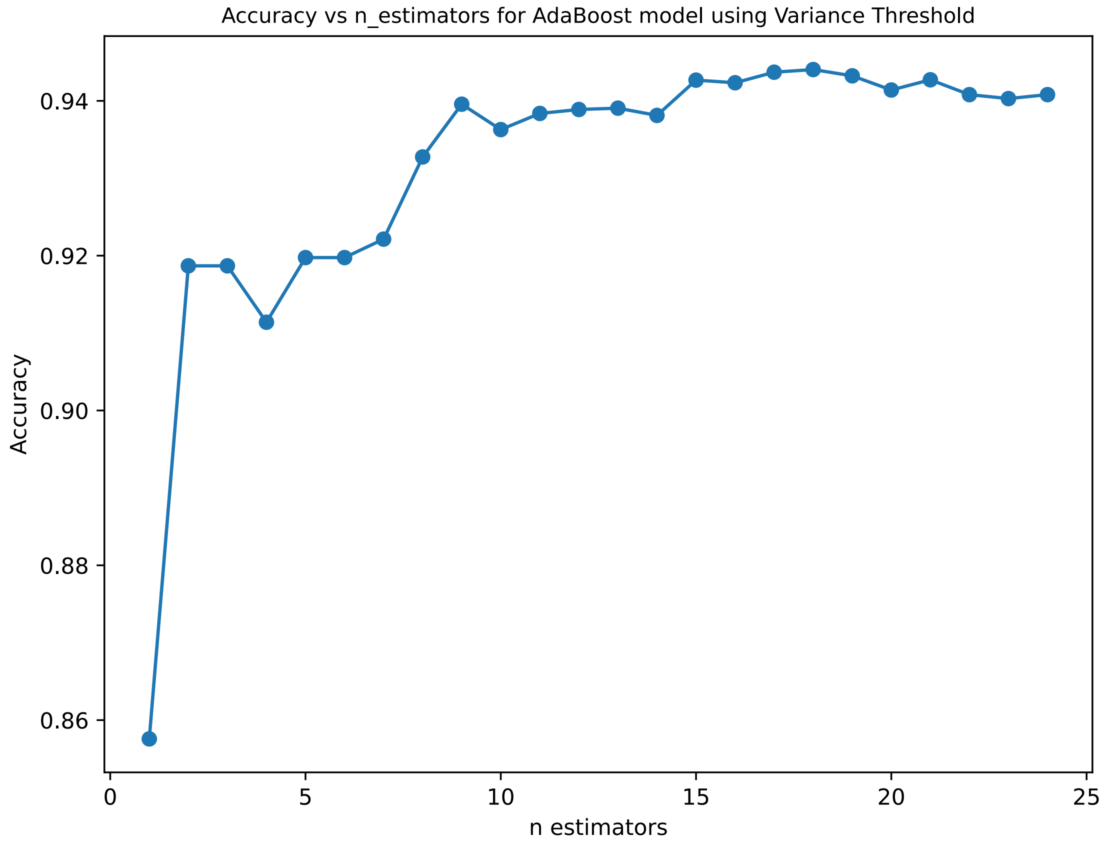
[21]:

```
#Adaboost hyperparameter tuning with variance threshold
abVar_pipe = Pipeline([('ab1','passthrough')])
abVar_params =[
               {'ab1':
 ↪[AdaBoostClassifier(random_state=100)],'ab1__n_estimators':range(25)}

              ]

cvStrat = StratifiedKFold(n_splits=10, shuffle=True, random_state=100)

abVar_grid = GridSearchCV(abVar_pipe, abVar_params, cv=cvStrat,␣
 ↪scoring=scoring,refit="Accuracy").fit(X_trainval_var,y_trainval_var)


# Report the best hyper-parameters and final test set performance
abVar_best_params = abVar_grid.best_params_

train_acc_abVar = metrics.accuracy_score(y_trainval_var,abVar_grid.
 ↪best_estimator_.predict(X_trainval_var))
test_acc_abVar = metrics.accuracy_score(y_test_var,abVar_grid.best_estimator_.
 ↪predict(X_test_var))
```

[22]:

```
#Adaboost hyperparameter tuning with varince threshold
n_estimators_list= range(25)
```

```
plt.figure(figsize=(8, 6))

plt.plot(n_estimators_list, abVar_grid.cv_results_['mean_test_Accuracy'][:
 ↪25],marker="o", label='Accuracy')
plt.xlabel('n estimators')
plt.ylabel('Accuracy')
plt.title('Accuracy vs n_estimators for AdaBoost model using Variance␣
 ↪Threshold',fontsize=9.5)
plt.show()
```

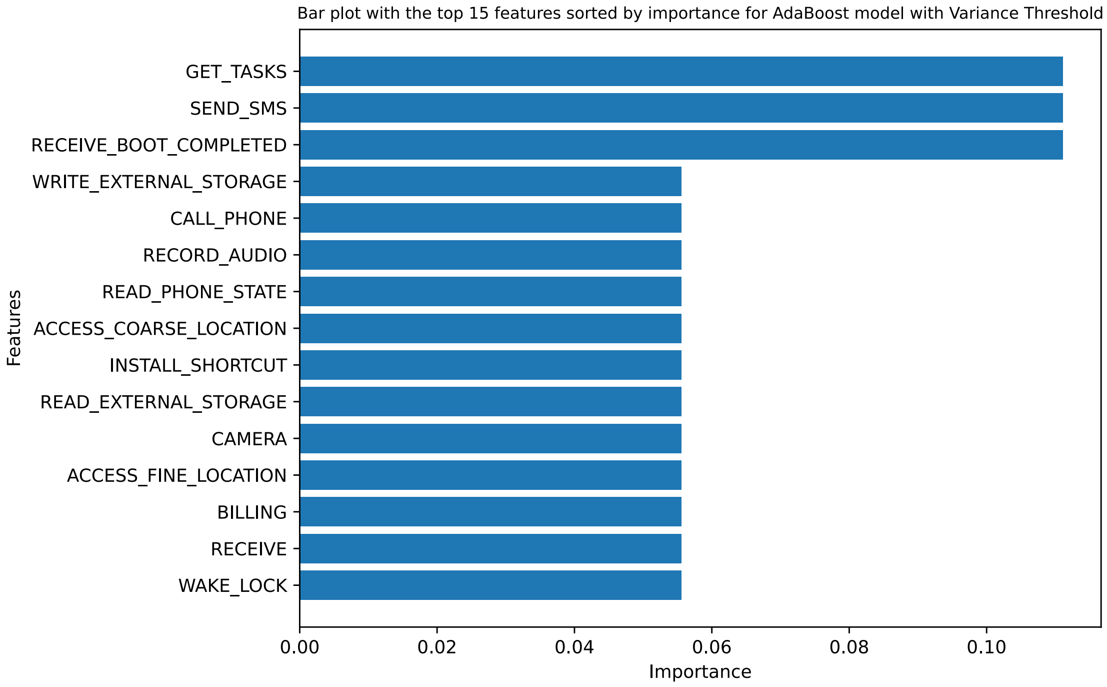


[23]:
```
print("Best paramters for Ada Boost model with Variance␣
 ↪Threshold",abVar_best_params,"\n")
print("Accuracy for Ada Boost model with Variance␣
 ↪Threshold",test_acc_abVar,train_acc_abVar)
```

```
Best paramters for Ada Boost model with Variance Threshold {'ab1':
AdaBoostClassifier(n_estimators=18, random_state=100), 'ab1__n_estimators': 18}

Accuracy for Ada Boost model with Variance Threshold 0.9423896369524459
0.9443000213083316
```

[24]:

```
abVar_importance = abVar_grid.best_estimator_.named_steps.ab1.
 ↪feature_importances_
abVar_imp = pd.DataFrame({'feature':cor_X.columns,'importance':
 ↪abVar_importance}).sort_values(by="importance", ascending=False)

# Create plot of top 15 features sorted by importance.
plt.figure(figsize=(8, 6))

plt.barh([i.split('.')[-1] for i in abVar_imp[:15].feature[::-1].
 ↪tolist()],abVar_imp[:15].importance[::-1])
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title('Bar plot with the top 15 features sorted by importance for AdaBoost␣
 ↪model with Variance Threshold',fontsize=9)
plt.show()
```



[25]:

```
#KNN hyperparameter tuning with variance threshold
knnVar_pipe = Pipeline([('knn1','passthrough')])
knnVar_params =[
               {'knn1':[neighbors.KNeighborsClassifier()],'knn1__n_neighbors':
 ↪range(15)}


              ]
```

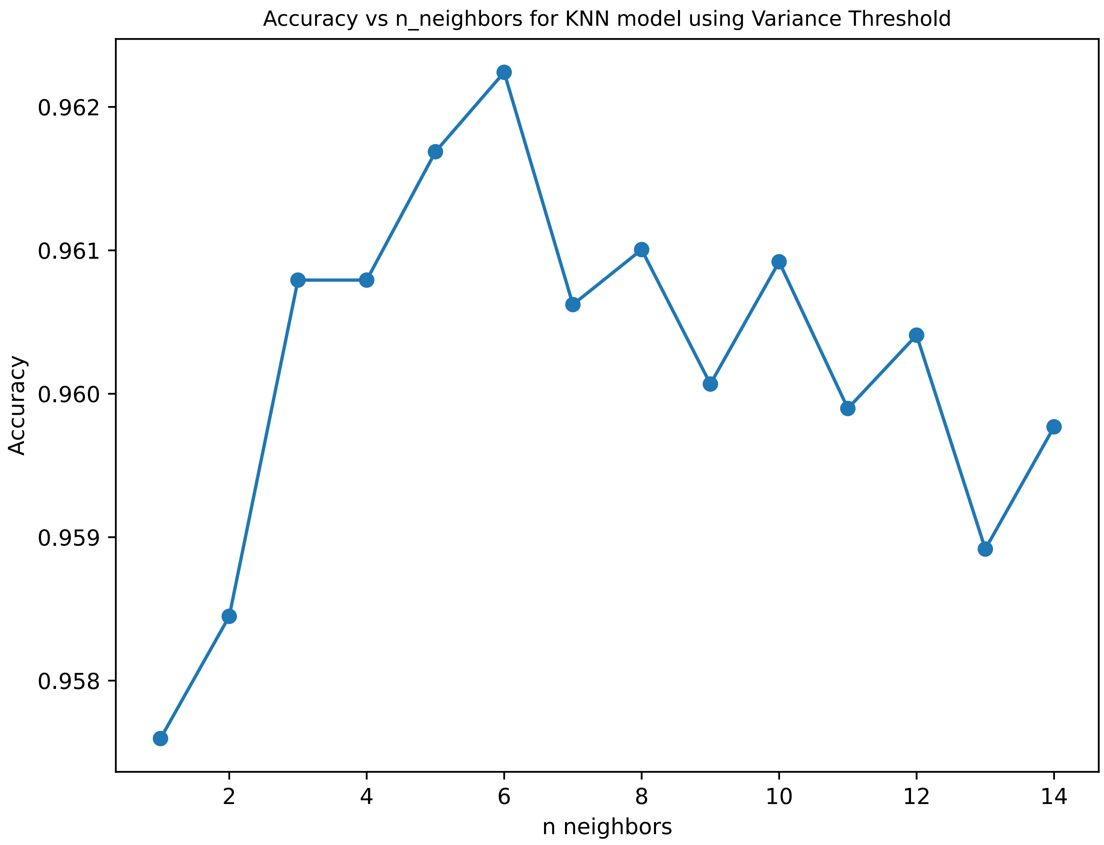
```
cvStrat = StratifiedKFold(n_splits=10, shuffle=True, random_state=100)

knnVar_grid = GridSearchCV(knnVar_pipe, knnVar_params, cv=cvStrat,␣
 ↪scoring=scoring,refit="Accuracy").fit(X_trainval_var.values,y_trainval_var)



# Report the best hyper-parameters and final test set performance
knnVar_best_params = knnVar_grid.best_params_
```

[26]:
```
train_acc_knnVar = metrics.accuracy_score(y_trainval_var,knnVar_grid.
 ↪best_estimator_.predict(X_trainval_var.values))
test_acc_knnVar = metrics.accuracy_score(y_test_var,knnVar_grid.best_estimator_.
 ↪predict(X_test_var.values))
```

[27]:
```
print("Best paramters for KNN model with Variance␣
 ↪Threshold",knnVar_best_params,"\n")
print("Accuracy for KNN model with Variance␣
 ↪Threshold",train_acc_knnVar,test_acc_knnVar)
```

```
Best paramters for KNN model with Variance Threshold {'knn1':
KNeighborsClassifier(n_neighbors=6), 'knn1__n_neighbors': 6}

Accuracy for KNN model with Variance Threshold 0.9683358193053484
0.9590932333390149
```

[28]:
```
#KNN tuning plot with variance threshold
n_estimators_list= range(15)
plt.figure(figsize=(8, 6))

plt.plot(n_estimators_list, knnVar_grid.cv_results_['mean_test_Accuracy'][:
 ↪15],marker="o", label='Train Accuracy')
plt.xlabel('n neighbors')
plt.ylabel('Accuracy')
plt.title('Accuracy vs n_neighbors for KNN model using Variance␣
 ↪Threshold',fontsize=9.5)
plt.show()
```



[29]:

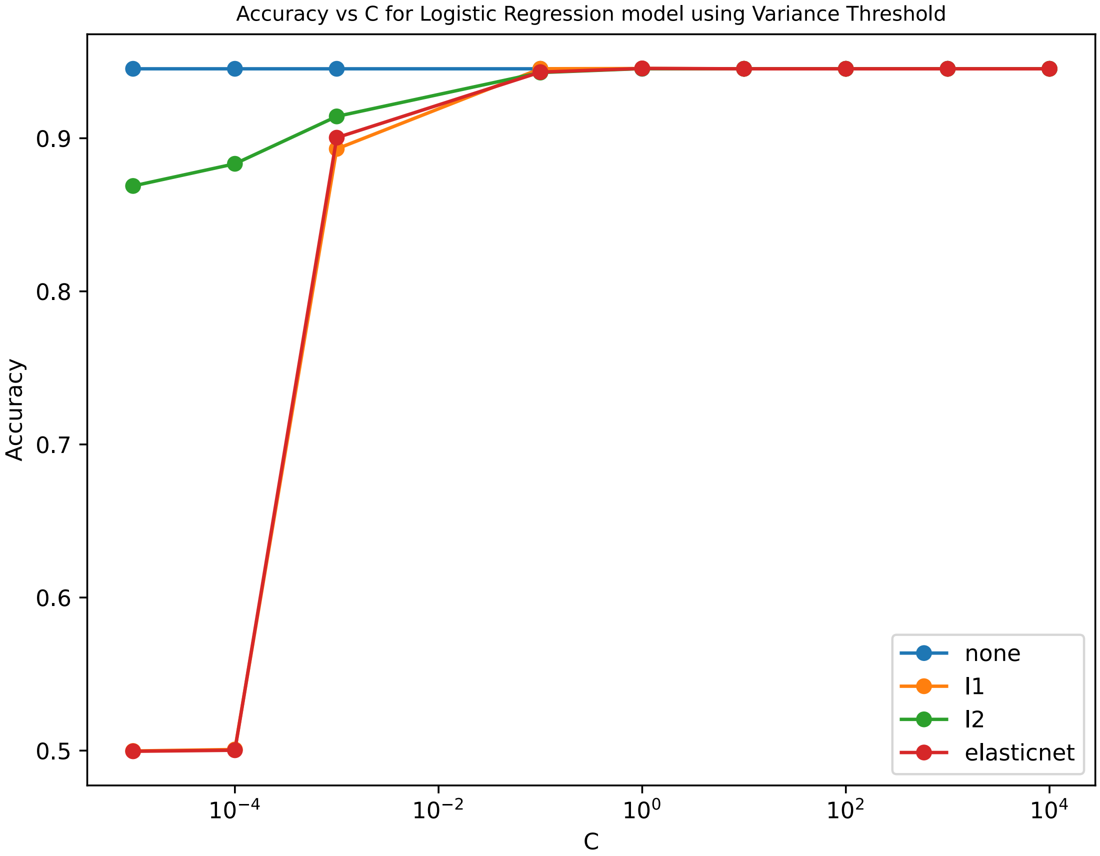
```
#Logistic Regression hyperparameter tuning with variance threshold
lrVar_pipe = Pipeline([('lr1','passthrough')])
lrVar_params =[
               {'lr1':[LogisticRegression()],'lr1__C':[0.00001,0.0001,0.001,0.
 ↪1, 1, 10, 100,1000,10000],
                'lr1__penalty': ['none','l1','l2','elasticnet'],'lr1__solver':
 ↪['saga'],'lr1__l1_ratio':[0.5]}

              ]

cvStrat = StratifiedKFold(n_splits=10, shuffle=True, random_state=100)

lrVar_grid = GridSearchCV(lrVar_pipe, lrVar_params, cv=cvStrat,␣
 ↪scoring=scoring,refit='Accuracy').fit(X_trainval_var,y_trainval_var)


# Report the best hyper-parameters and final test set performance
lrVar_best_params = lrVar_grid.best_params_
```

```
train_acc_lrVar = metrics.accuracy_score(y_trainval_var,lrVar_grid.
 ↪best_estimator_.predict(X_trainval_var))
test_acc_lrVar = metrics.accuracy_score(y_test_var,lrVar_grid.best_estimator_.
 ↪predict(X_test_var))
```

```
[30]: print('Logistic Regression best parameters with chi2 :',lrVar_best_params)
      print('Logistic Regression accuracy with chi2 :',train_acc_lrVar,test_acc_lrVar)
```

```
Logistic Regression best parameters with chi2 : {'lr1': LogisticRegression(C=1,
l1_ratio=0.5, penalty='elasticnet', solver='saga'), 'lr1__C': 1,
'lr1__l1_ratio': 0.5, 'lr1__penalty': 'elasticnet', 'lr1__solver': 'saga'}
Logistic Regression accuracy with chi2 : 0.9456637545280204 0.9463098687574569
```

```
[31]: n_estimators_list= [0.00001,0.0001,0.001,0.1, 1, 10, 100,1000,10000]
      plt.figure(figsize=(8, 6))

      #Logistic Regression tuning plot with variance threshold

      plt.plot(n_estimators_list, [lrVar_grid.cv_results_['mean_test_Accuracy'][i]␣
       ↪for i in range(36) if i%4==0],marker="o", label='Train Accuracy')
      plt.plot(n_estimators_list, [lrVar_grid.cv_results_['mean_test_Accuracy'][i]␣
       ↪for i in range(36) if i%4==1],marker="o", label='Train Accuracy')
      plt.plot(n_estimators_list, [lrVar_grid.cv_results_['mean_test_Accuracy'][i]␣
       ↪for i in range(36) if i%4==2],marker="o", label='Train Accuracy')
      plt.plot(n_estimators_list, [lrVar_grid.cv_results_['mean_test_Accuracy'][i]␣
       ↪for i in range(36) if i%4==3],marker="o", label='Train Accuracy')
      plt.legend(['none','l1','l2','elasticnet'])
      plt.xscale('log')
      plt.xlabel('C')
      plt.ylabel('Accuracy')
      plt.title('Accuracy vs C for Logistic Regression model using Variance␣
       ↪Threshold',fontsize=9)
      plt.show()
```



0.4 Chi-Square feature selection technique

[32]:

```
#Chi-Square feature selection with k=25
chi2_selector = SelectKBest(score_func=chi2, k=25)

chi2_X= chi2_selector.fit_transform(X,y)
```

[33]:

```
# Split of the test set
X_trainval_chi2, X_test_chi2, y_trainval_chi2, y_test_chi2 =␣
 ↪train_test_split(chi2_X, y, test_size=0.2, stratify=y, random_state=100)

# Split trainval into train + val
X_train_chi2, X_val_chi2, y_train_chi2, y_val_chi2 =␣
 ↪train_test_split(X_trainval_chi2, y_trainval_chi2, test_size=0.25,␣
 ↪stratify=y_trainval_chi2, random_state=100)
```

[34]:

```
chi2_indices = chi2_selector.get_support(indices=True)
```

```
# Get the feature names using the selected indices
chi2_features = X.columns[chi2_indices]
```

[35]:
```
#Setting scoring metrics
scoring = {
    'AUC': 'roc_auc',
    'Accuracy': metrics.make_scorer(metrics.accuracy_score)
}
```

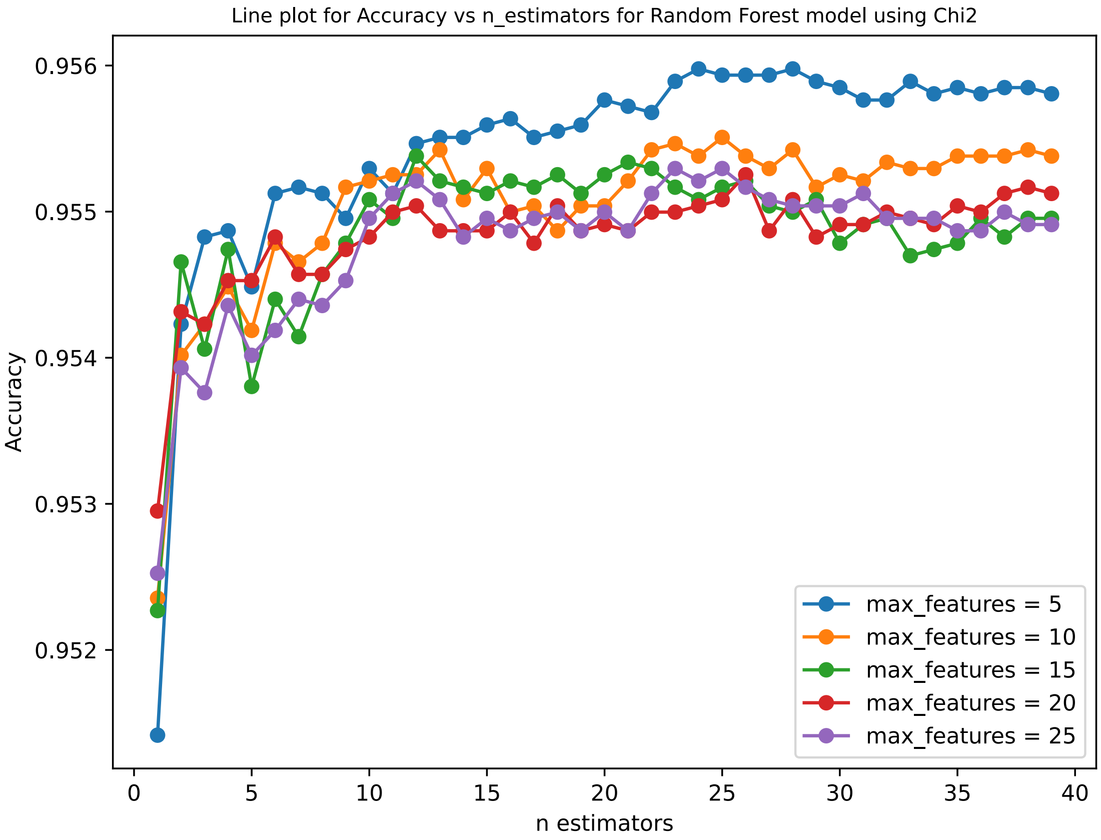
[36]:
```
#Random forest hyperparameter tuning with Chi-Square
rfChi2_pipe = Pipeline([('rf2','passthrough')])
rfChi2_params =[
               {'rf2':
 ↪[RandomForestClassifier(random_state=100)],'rf2__n_estimators':range(40),
                'rf2__max_features':[5,10,15,20,25]}

              ]
cvStrat = StratifiedKFold(n_splits=10, shuffle=True, random_state=100)


rfChi2_grid = GridSearchCV(rfChi2_pipe, rfChi2_params, cv=cvStrat,␣
 ↪scoring=scoring,refit="Accuracy").fit(X_trainval_chi2,y_trainval_chi2)


# Report the best hyper-parameters and final test set performance
rfChi2_best_params = rfChi2_grid.best_params_

train_acc_rfChi2 = metrics.accuracy_score(y_trainval_chi2,rfChi2_grid.
 ↪best_estimator_.predict(X_trainval_chi2))
test_acc_rfChi2 = metrics.accuracy_score(y_test_chi2,rfChi2_grid.
 ↪best_estimator_.predict(X_test_chi2))
```

[37]:
```
print('Random forest best parameters with chi2 :',rfChi2_best_params)
print('Random forest Accuracy with chi2 :',train_acc_rfChi2,test_acc_rfChi2)
```

```
Random forest best parameters with chi2 : {'rf2':
RandomForestClassifier(max_features=5, n_estimators=28, random_state=100),
'rf2__max_features': 5, 'rf2__n_estimators': 28}
Random forest Accuracy with chi2 : 0.962753036437247 0.954491222089654
```

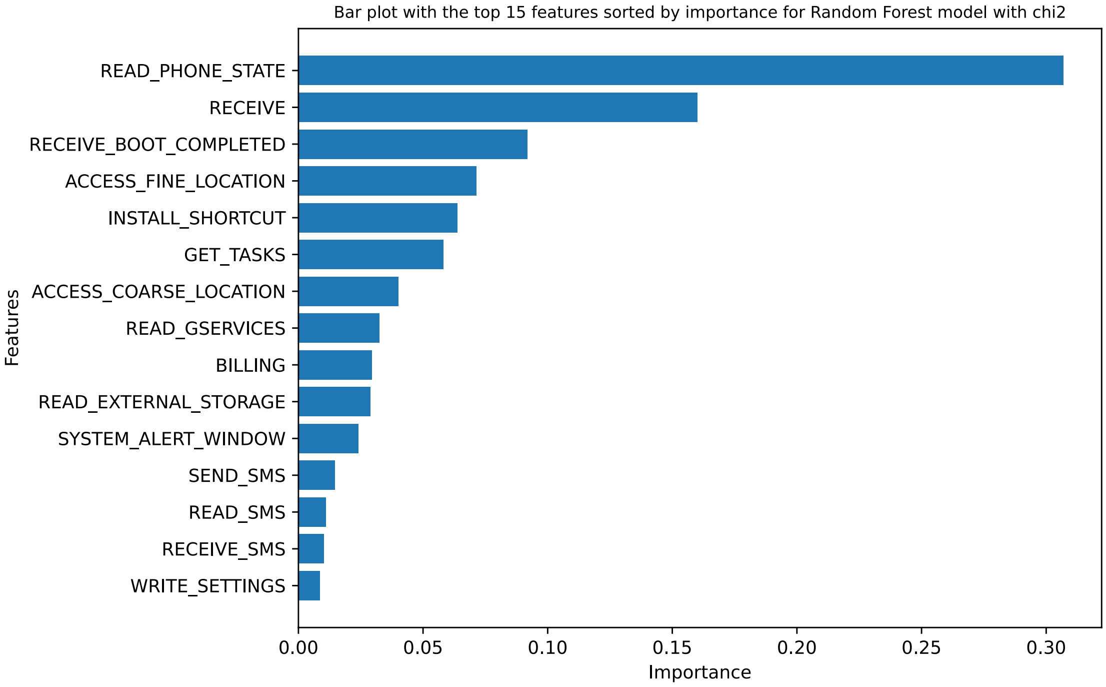
[38]:
```
#Random forest tuning plot with Chi-Square
n_estimators_list= range(40)
plt.figure(figsize=(8, 6))

plt.plot(n_estimators_list, rfChi2_grid.cv_results_['mean_test_Accuracy'][:
 ↪40],marker="o", label='Train Accuracy')
```

```
plt.plot(n_estimators_list, rfChi2_grid.cv_results_['mean_test_Accuracy'][40:
 ↪80],marker="o", label='Train Accuracy')
plt.plot(n_estimators_list, rfChi2_grid.cv_results_['mean_test_Accuracy'][80:
 ↪120],marker="o", label='Train Accuracy')
plt.plot(n_estimators_list, rfChi2_grid.cv_results_['mean_test_Accuracy'][120:
 ↪160],marker="o", label='Train Accuracy')
plt.plot(n_estimators_list, rfChi2_grid.cv_results_['mean_test_Accuracy'][160:
 ↪200],marker="o", label='Train Accuracy')
plt.legend(['max_features = 5','max_features = 10','max_features =␣
 ↪15','max_features = 20','max_features = 25'])

plt.xlabel('n estimators')
plt.ylabel('Accuracy')
plt.title('Line plot for Accuracy vs n_estimators for Random Forest model using␣
 ↪Chi2',fontsize=9)
plt.show()
```



[39]:
```
# Calculate performance on test data `auc_test`
rfChi2_auc_test = metrics.roc_auc_score(y_test_chi2,rfChi2_grid.best_estimator_.
 ↪predict(X_test_chi2))
```

```
rfChi2_importance = rfChi2_grid.best_estimator_.named_steps.rf2.
 ↪feature_importances_
rfChi2_imp = pd.DataFrame({'feature':chi2_features,'importance':
 ↪rfChi2_importance}).sort_values(by="importance", ascending=False)
# Create plot of top 15 features sorted by importance.
plt.figure(figsize=(8, 6))

plt.barh([i.split('.')[-1] for i in rfChi2_imp[:15].feature[::-1].
 ↪tolist()],rfChi2_imp[:15].importance[::-1])
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title('Bar plot with the top 15 features sorted by importance for Random␣
 ↪Forest model with chi2',fontsize=9)
plt.show()
```



[40]:
```
#Adaboost hyperparameter tuning with Chi-Square
abChi2_pipe = Pipeline([('ab2','passthrough')])
abChi2_params =[
               {'ab2':
 ↪[AdaBoostClassifier(random_state=100)],'ab2__n_estimators':range(25)}

              ]
```

```
cvStrat = StratifiedKFold(n_splits=10, shuffle=True, random_state=100)

abChi2_grid = GridSearchCV(abChi2_pipe, abChi2_params, cv=cvStrat,␣
 ↪scoring=scoring,refit="Accuracy").fit(X_trainval_chi2,y_trainval_chi2)


# Report the best hyper-parameters and final test set performance
abChi2_best_params = abChi2_grid.best_params_

train_acc_abChi2 = metrics.accuracy_score(y_trainval_chi2,abChi2_grid.
 ↪best_estimator_.predict(X_trainval_chi2))
test_acc_abChi2 = metrics.accuracy_score(y_test_chi2,abChi2_grid.
 ↪best_estimator_.predict(X_test_chi2))
```

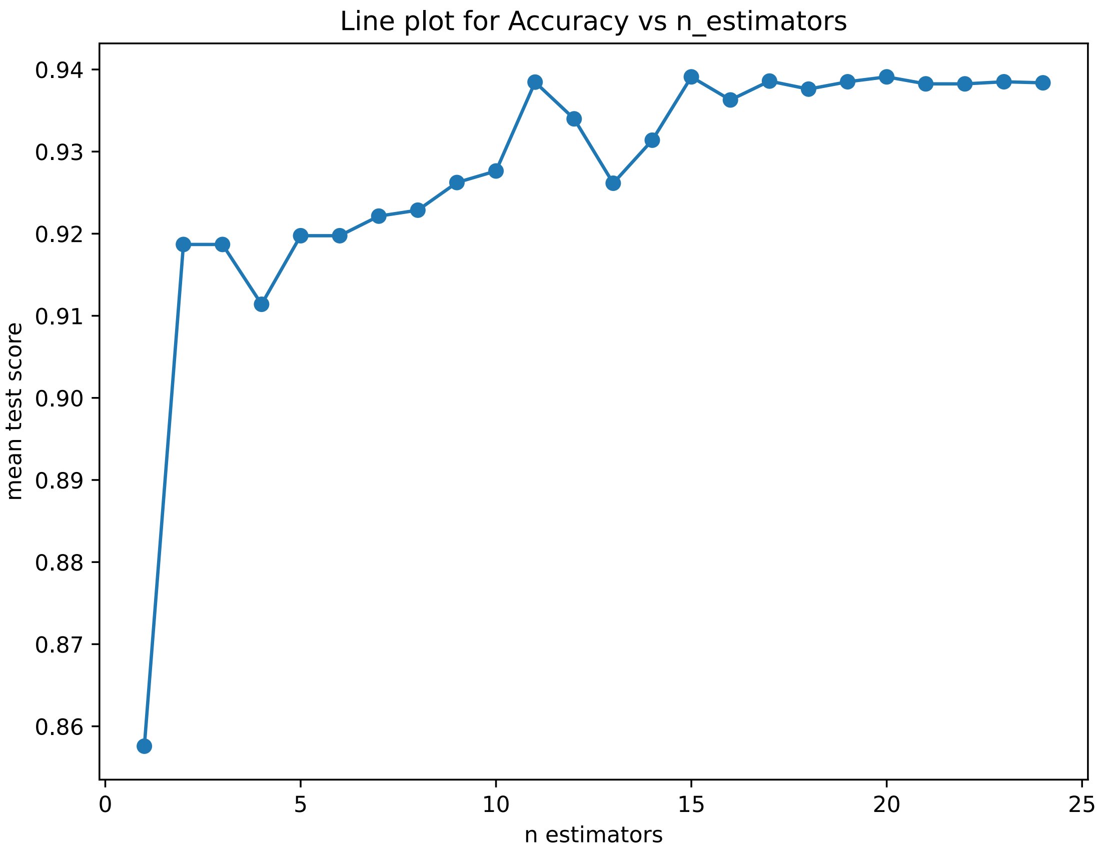
[41]:

```
print('Ada Boost best parameters with chi2 :',abChi2_best_params)
print('Ada Boost AUC with chi2 :',train_acc_abChi2,test_acc_abChi2)
```

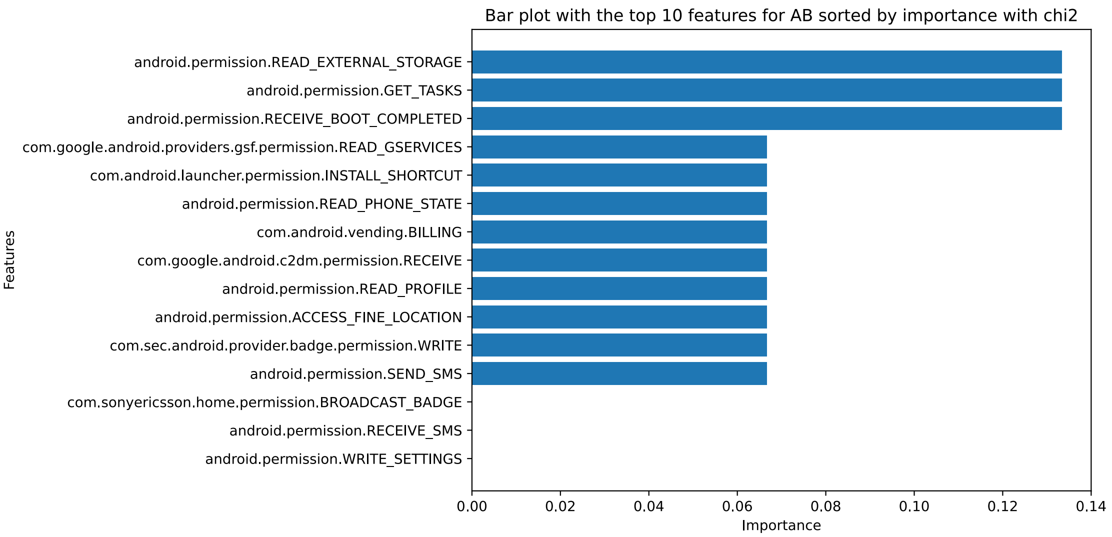
```
Ada Boost best parameters with chi2 : {'ab2':
AdaBoostClassifier(n_estimators=15, random_state=100), 'ab2__n_estimators': 15}
Ada Boost AUC with chi2 : 0.9402514383123801 0.9396625191750468
```

[46]:

```
#Adaboost hyperparameter tuning plot with Chi-Square
n_estimators_list= range(25)
plt.figure(figsize=(8, 6))

plt.plot(n_estimators_list, abChi2_grid.cv_results_['mean_test_Accuracy'][:
 ↪25],marker="o", label='Train Accuracy')
plt.xlabel('n estimators')
plt.ylabel('mean test score')
plt.title('Line plot for Accuracy vs n_estimators')
plt.show()
```



[47]:
```
abChi2_importance = abChi2_grid.best_estimator_.named_steps.ab2.
  ↪feature_importances_
abChi2_imp = pd.DataFrame({'feature':chi2_features,'importance':
  ↪abChi2_importance}).sort_values(by="importance", ascending=False)
# Create plot of top 15 features sorted by importance.
plt.figure(figsize=(8, 6))

plt.barh(abChi2_imp[:15].feature[::-1],abChi2_imp[:15].importance[::-1])
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title('Bar plot with the top 10 features for AB sorted by importance with␣
  ↪chi2')
plt.show()
```



[48]:

```
#KNN hyperparameter tuning with Chi-Square
= Pipeline([('knn2','passthrough')])
knnChi2_params =[
               {'knn2':[neighbors.KNeighborsClassifier()],'knn2__n_neighbors':
  ↪range(20)}

              ]

cvStrat = StratifiedKFold(n_splits=10, shuffle=True, random_state=100)

knnChi2_grid = GridSearchCV(knnChi2_pipe, knnChi2_params, cv=cvStrat,␣
  ↪scoring=scoring,refit="Accuracy").fit(X_trainval_chi2,y_trainval_chi2)


# Report the best hyper-parameters and final test set performance
knnChi2_best_params = knnChi2_grid.best_params_

train_acc_knnChi2 = metrics.accuracy_score(y_trainval_chi2,knnChi2_grid.
  ↪best_estimator_.predict(X_trainval_chi2))
test_acc_knnChi2 = metrics.accuracy_score(y_test_chi2,knnChi2_grid.
  ↪best_estimator_.predict(X_test_chi2))
```

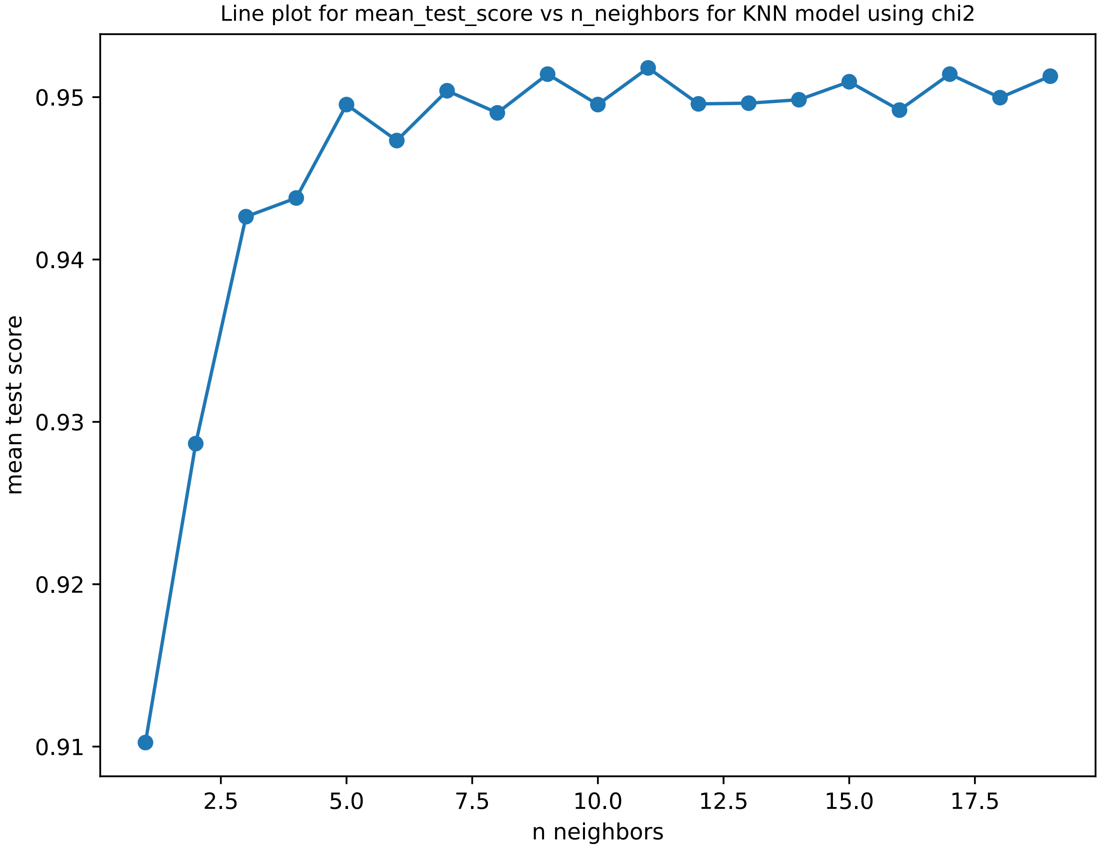
[49]:

```
print('KNN best parameters with chi2 :',knnChi2_best_params)
print('KNN Aaccuracy with chi2 :',train_acc_knnChi2,test_acc_knnChi2)
```

```
KNN best parameters with chi2 : {'knn2': KNeighborsClassifier(n_neighbors=11),
'knn2__n_neighbors': 11}
KNN Aaccuracy with chi2 : 0.9545280204559983 0.9493778762570309
```

[50]:

```
#KNN tuning plot with Chi-Square
n_estimators_list= range(20)
plt.figure(figsize=(8, 6))

plt.plot(n_estimators_list, knnChi2_grid.cv_results_['mean_test_Accuracy'][:
 ↪20],marker="o", label='Train Accuracy')
plt.xlabel('n neighbors')
plt.ylabel('mean test score')
plt.title('Line plot for mean_test_score vs n_neighbors for KNN model using␣
 ↪chi2',fontsize=9.5)
plt.show()
```



[51]:

```
#Logistic Regression hyperparameter tuning with Chi-Square
lrChi2_pipe = Pipeline([('lr2','passthrough')])
lrChi2_params =[
               {'lr2':[LogisticRegression()],'lr2__C':[0.00001,0.0001,0.001,0.
 ↪1, 1, 10, 100,1000,10000],
                'lr2__penalty': ['none','l1','l2','elasticnet'],'lr2__solver':
 ↪['saga'],'lr2__l1_ratio':[0.5]}
```

```
              ]

cvStrat = StratifiedKFold(n_splits=10, shuffle=True, random_state=100)

lrChi2_grid = GridSearchCV(lrChi2_pipe, lrChi2_params, cv=cvStrat,␣
 ↪scoring=scoring,refit='Accuracy').fit(X_trainval_chi2,y_trainval_chi2)



# Report the best hyper-parameters and final test set performance
lrChi2_best_params = lrChi2_grid.best_params_

train_acc_lrChi2 = metrics.accuracy_score(y_trainval_chi2,lrChi2_grid.
 ↪best_estimator_.predict(X_trainval_chi2))
test_acc_lrChi2 = metrics.accuracy_score(y_test_chi2,lrChi2_grid.
 ↪best_estimator_.predict(X_test_chi2))
```

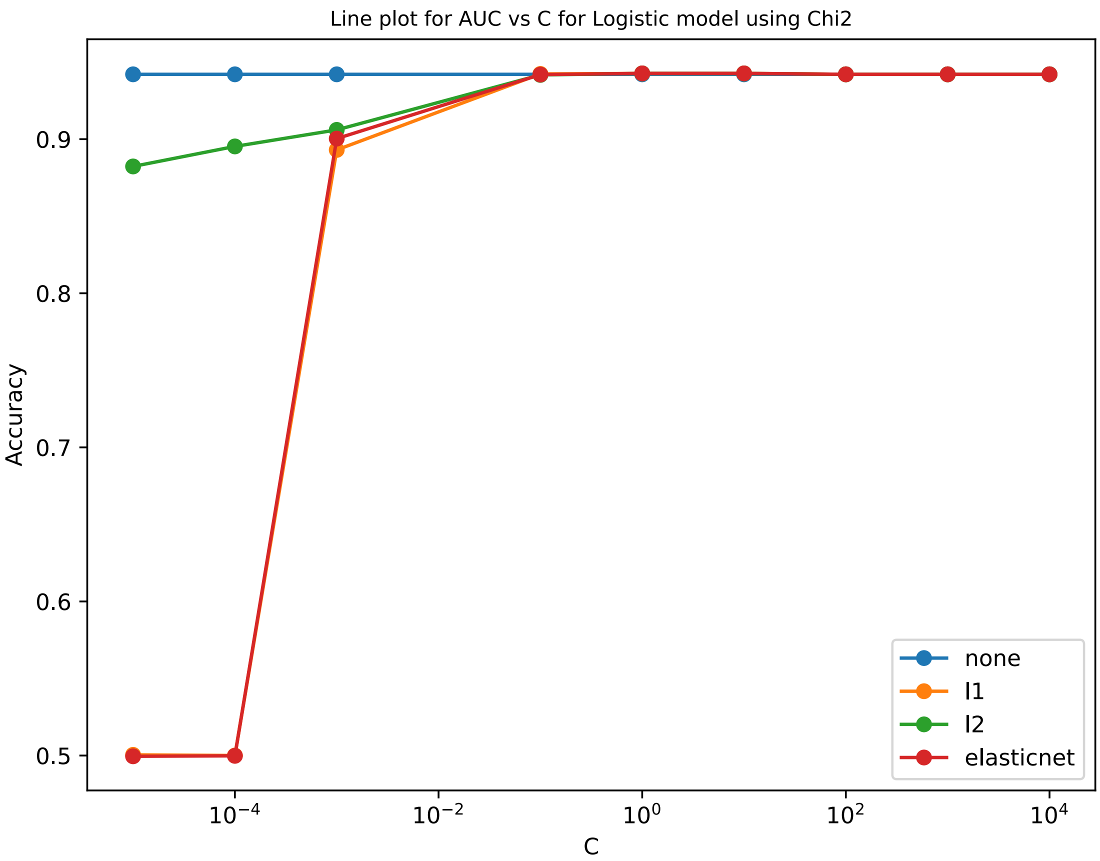
[52]:

```
print('Logistic Regression best parameters with chi2 :',lrChi2_best_params)
print('Logistic Regression Aaccuracy with chi2 :
 ↪',train_acc_lrChi2,test_acc_lrChi2)
```

```
Logistic Regression best parameters with chi2 : {'lr2': LogisticRegression(C=1,
l1_ratio=0.5, solver='saga'), 'lr2__C': 1, 'lr2__l1_ratio': 0.5, 'lr2__penalty':
'l2', 'lr2__solver': 'saga'}
Logistic Regression Aaccuracy with chi2 : 0.9438738546771788 0.9425600818135333
```

[53]:

```
#Logistic Regression tuning plot with Chi-Square
n_estimators_list= [0.00001,0.0001,0.001,0.1, 1, 10, 100,1000,10000]
plt.figure(figsize=(8, 6))

plt.xscale('log')
plt.plot(n_estimators_list, [lrChi2_grid.cv_results_['mean_test_Accuracy'][i]␣
 ↪for i in range(36) if i%4==0],marker="o", label='Train Accuracy')
plt.plot(n_estimators_list, [lrChi2_grid.cv_results_['mean_test_Accuracy'][i]␣
 ↪for i in range(36) if i%4==1],marker="o", label='Train Accuracy')
plt.plot(n_estimators_list, [lrChi2_grid.cv_results_['mean_test_Accuracy'][i]␣
 ↪for i in range(36) if i%4==2],marker="o", label='Train Accuracy')
plt.plot(n_estimators_list, [lrChi2_grid.cv_results_['mean_test_Accuracy'][i]␣
 ↪for i in range(36) if i%4==3],marker="o", label='Train Accuracy')
plt.legend(['none','l1','l2','elasticnet'])

plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title('Line plot for AUC vs C for Logistic model using Chi2',fontsize=9)
plt.show()
```



1 Metrics for best models from both the feature selection techniques

1.1 With Variance threshold

1.1.1 Random Forest

[54]:

```
#Calculating metrics precesion, recall, f1 score
clf = RandomForestClassifier(random_state=100,n_estimators=47,max_features=5)
clf.fit(X_trainval_var, y_trainval_var)

y_pred_var = clf.predict(X_test_var)

precision_var = precision_score(y_test_var, y_pred_var)
recall_var = recall_score(y_test_var, y_pred_var)
f1_var = f1_score(y_test_var, y_pred_var)
```

```
accuracy_var = metrics.accuracy_score(y_test_var, y_pred_var)

print("Accuracy:", round(accuracy_var,4))
print("Precision:", round(precision_var,4))
print("Recall:", round(recall_var,4))
print("F1 Score:", round(f1_var,4))
```

```
Accuracy: 0.9618
Precision: 0.9638
Recall: 0.9599
F1 Score: 0.9618
```

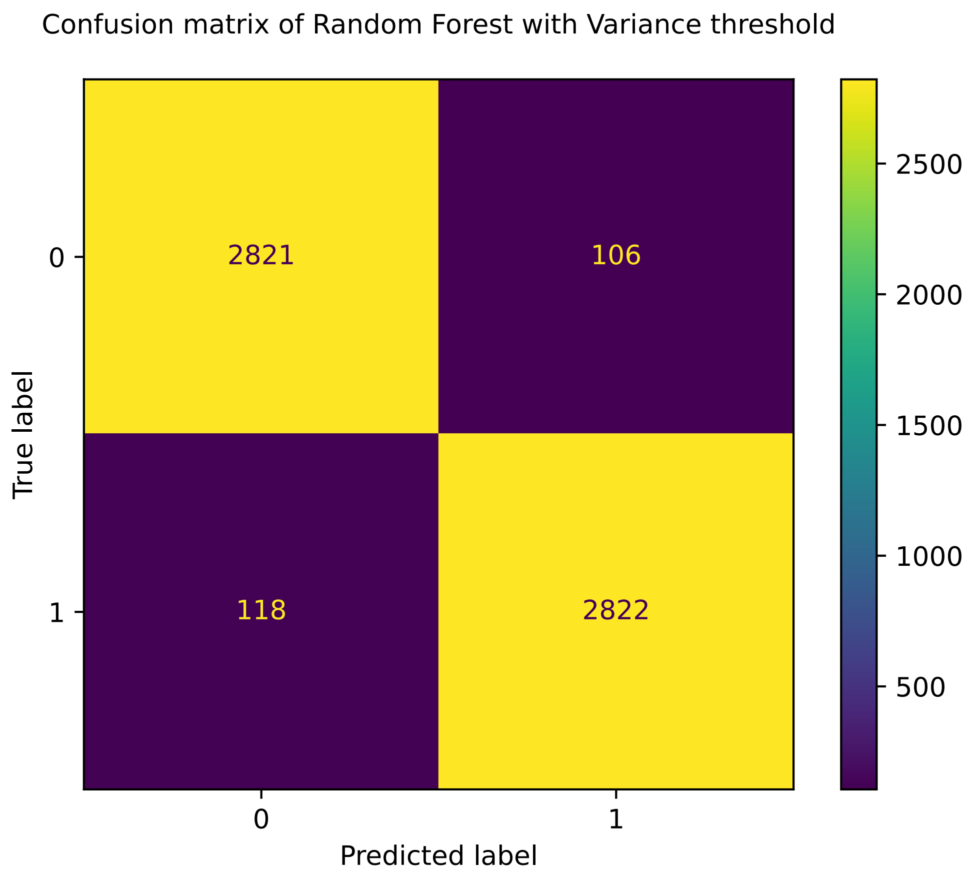
[55]:

```
#Calulating TPR and FPR
cm_var = confusion_matrix(y_test_var,y_pred_var)
TN,FP,FN,TP=cm_var.ravel()
tpr=TP/(TP+FN)
fpr=FP/(FP+TN)
print("True Positive Rate :",round(tpr,4),"\n"+"False Positive Rate :
 ↪",round(fpr,4))
```

```
True Positive Rate : 0.9599
False Positive Rate : 0.0362
```

[56]:

```
#Confusion matrix for Random Forest model with Variance threshold
ConfusionMatrixDisplay(confusion_matrix=cm_var).plot()
plt.title("Confusion matrix of Random Forest with Variance␣
 ↪threshold\n",fontsize=10)
plt.show()
```



1.2 With Chi-Square Statstic

1.3 Random Forest

[58]:

```
#Calculating metrics precesion, recall, f1 score
clf = RandomForestClassifier(random_state=100,n_estimators=28,max_features=5)
clf.fit(X_trainval_chi2, y_trainval_chi2)

y_pred_chi2 = clf.predict(X_test_chi2)

precision = precision_score(y_test_chi2, y_pred_chi2)
recall = recall_score(y_test_chi2, y_pred_chi2)
f1 = f1_score(y_test_chi2, y_pred_chi2)

accuracy = metrics.accuracy_score(y_test_chi2, y_pred_chi2)

print("F1 Score:", round(f1,4))
print("Precision:", round(precision,4))
print("Recall:", round(recall,4))
print("Accuracy:", round(accuracy,4))
```

```
F1 Score: 0.955
Precision: 0.9462
Recall: 0.9639
Accuracy: 0.9545
```

[59]:

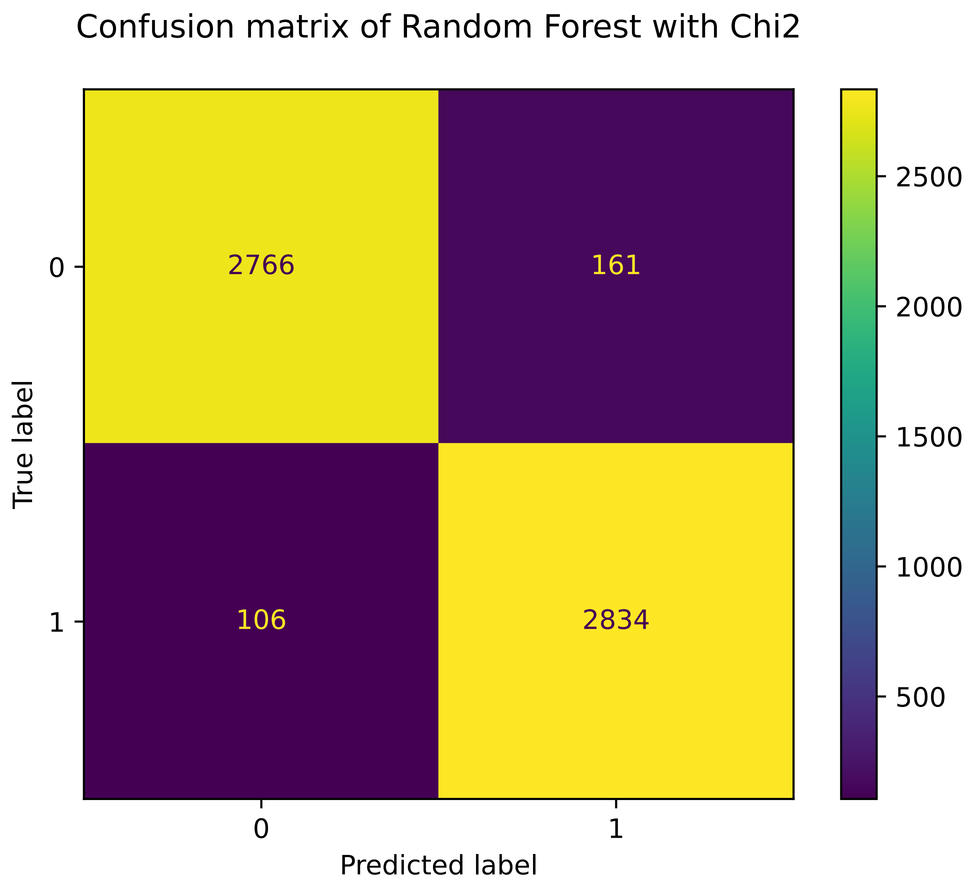
```
#Calulating TPR and FPR
cm_chi2 = confusion_matrix(y_test_var,y_pred_chi2)
TN,FP,FN,TP=cm_chi2.ravel()
tpr=TP/(TP+FN)
fpr=FP/(FP+TN)
print("True Positive Rate :",round(tpr,4),"\n"+"False Positive Rate :
  ↪",round(fpr,4))
```

```
True Positive Rate : 0.9639
False Positive Rate : 0.055
```

1.3.1 Confusion Matrix

[61]:

```
#Confusion matrix for Random Forest model with Chi-Square statistic
ConfusionMatrixDisplay(confusion_matrix=cm_chi2).plot()
plt.title("Confusion matrix of Random Forest with Chi2\n",fontsize=12)
plt.savefig("test.png")
```


Confusion matrix of Random Forest with Chi2
True label
0
1
2766
161
106
2834
Predicted label
0
1
2500
2000
1500
1000
500