Classification Algorithms

June 30, 2022

1 Classification Algorithms

Imports and Configurations for thge project

[69]:
```
import matplotlib.pyplot as plt
from sklearn import metrics
import seaborn as sns
import pandas as pd
import numpy as np

import warnings

warnings.simplefilter(action='ignore', category=FutureWarning)
plt.style.use('seaborn-deep')
```

Load training data uding pandas

[70]:
```
# load adult train data
train_data =  pd.read_csv("adult.data", header=None)
```

[71]:
```
# assign columns
train_data.columns =␣
 ↪['age','workclass','fnlwgt','education','education-num','marital-status',
'occupation','relationship','race','sex','capital-gain','capital-loss','hours-per-week','nativ
```

[72]:
```
# remove rows with unknown '?'
#print("train data with '?:",train_data[(train_data == ' ?')].count(axis=1).
 ↪sum())
train_data = train_data[~(train_data == ' ?').any(axis=1)]
#print("after cleaning:",train_data[(train_data == ' ?')].count(axis=1).sum())
```

[73]:
```
# reindex rows
train_data.reset_index(inplace=True);

# print train data
#print(train_data)
#print("train data records: ",len(train_data))
```

Load test data

[74]:
```
# load adult test data
test_data =  pd.read_csv("adult.test", header=None)
#print(test_data)
```

[75]:
```
# assign columns
test_data.columns =␣
 ↪['age','workclass','fnlwgt','education','education-num','marital-status',
'occupation','relationship','race','sex','capital-gain','capital-loss','hours-per-week','nativ
#print(test_data)
```

[76]:
```
# remove rows with unknown '?'
#print("train data with '?:",test_data[(test_data == ' ?')].count(axis=0).sum())
test_data = test_data[~(test_data == ' ?').any(axis=1)]
#print("after cleaning:",test_data[(test_data == ' ?')].count(axis=0).sum())
```

[77]:
```
# reindex rows
test_data.reset_index(inplace=True);

# print train data
#print(test_data)
#print("number testdata records: ",len(test_data))
```

Importing Sklearn's encoding modules to perform the One-hot encoding

[78]:
```
# apply one-hot encoding to transform data columns
#␣
 ↪workclass,education,marital-status,occupation,relationship,race,sex,native-country
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder

# make encodees
labelEncoder = LabelEncoder()
oneHotEncoder = OneHotEncoder(handle_unknown='ignore')
```

Encoding train set

[79]:
```
###################
# encode train set #
###################

#print("\nencode train set")

# encode sex column
#print("encoding column: sex")
train_data['sex-code'] = labelEncoder.fit_transform(train_data['sex'])
```

```
# drop sex table
train_data.drop('sex', axis=1, inplace=True)

# encode native-country column
#print("encoding column: native-country")
train_data['native-country-code'] = labelEncoder.
 ↪fit_transform(train_data['native-country'])

# drop native-country table
train_data.drop('native-country', axis=1, inplace=True)


# list of columns to encode for one-hot-decoding
encode_columns =␣
 ↪['workclass','education','marital-status','occupation','relationship','race']


# for each column
for column in encode_columns:

    # print column
    #print("one-hot-encoding column: " + column)

    # encode workclass column
    encoding = oneHotEncoder.fit_transform(train_data[[column]])
    # encoding as an array
    encoding = encoding.toarray()
    # make a data frame of encoded results
    encoded = pd.DataFrame(encoding)
    # get encoded column names
    encoded_columns = oneHotEncoder.get_feature_names_out([column])
    # assign encoded column names
    encoded.columns=encoded_columns
    # add encoded columns to original  dataframe
    train_data = train_data.join(encoded)
    # drop original columnh
    train_data.drop(column, axis=1, inplace=True)
```

Encoding test set

[80]:
```
###################
# encode test set #
###################

#print("\nencode test set")
```

```
# encode sex column
#print("encoding column: sex")
test_data['sex-code'] = labelEncoder.fit_transform(test_data['sex'])

# drop sex table
test_data.drop('sex', axis=1, inplace=True)

# encode native-country column
#print("encoding column: native-country")
test_data['native-country-code'] = labelEncoder.
 ↪fit_transform(test_data['native-country'])

# drop native-country table
test_data.drop('native-country', axis=1, inplace=True)


# list of columns to encode to one-hot-decoding
encode_columns =␣
 ↪['workclass','education','marital-status','occupation','relationship','race']

# for each column
for column in encode_columns:


    # print column
    #print("one-hot-encoding column: " + column)

    # encode workclass column
    encoding = oneHotEncoder.fit_transform(test_data[[column]])
    # encoding as an array
    encoding = encoding.toarray()
    # make a data frame of encoded results
    encoded = pd.DataFrame(encoding)
    # get encoded column names
    encoded_columns = oneHotEncoder.get_feature_names_out([column])
    # assign encoded column names
    encoded.columns=encoded_columns
    # add encoded columns to original  dataframe
    test_data = test_data.join(encoded)
    # drop original columnh
    test_data.drop(column, axis=1, inplace=True)
```

Defining the features and attributes from data for the Model inputs

[81]:
```
# train data X,y
X_train = train_data[train_data.columns.drop('salary')]
y_train = train_data['salary']
```

```
# test data  X,y
X_test = test_data[test_data.columns.drop('salary')]
y_test = test_data['salary']
```

2 Models

DT Classifier Import sklearns decision tree algorithm. Create a class object (dt) from the class algorithm and train it (dt.fit(…)) using the features and attributes (X_train, y_train) from the data

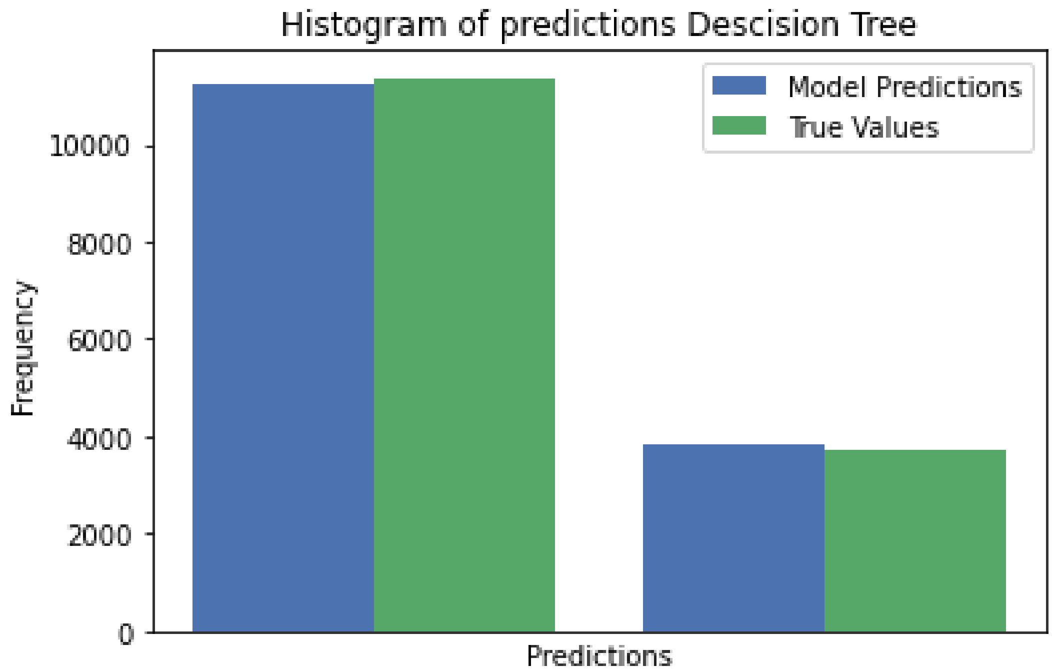
[ ]:
```
#############################
# descision tree classifier #
#############################

from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


#print("\nDescision Tree classifier")

# make decision tree Classifier
dt = tree.DecisionTreeClassifier()

# fit data
dt.fit(X_train, y_train)
```

Using the dt object created to predict the test features (X_test)

[83]:
```
# calculte prediction
dt_predicted = dt.predict(X_test);
#print( predicted)
```

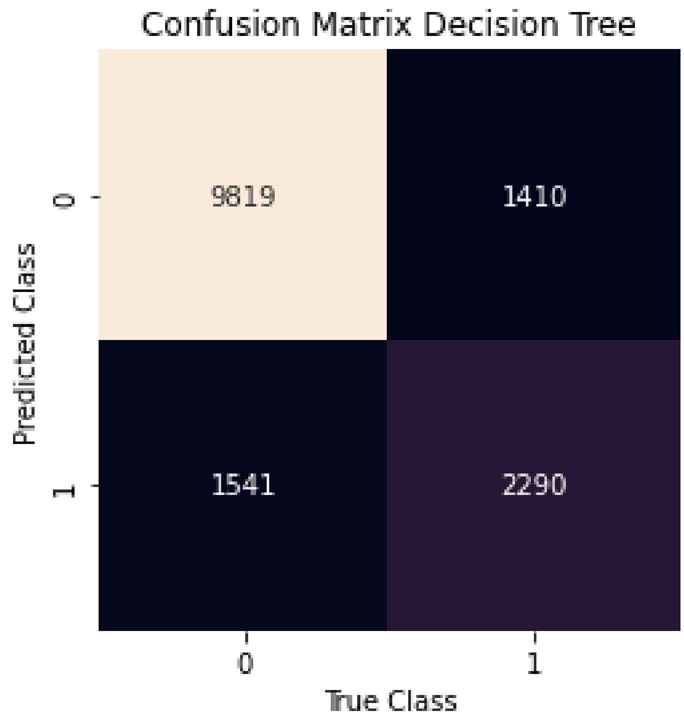
Plotting Histogram of predictions against true values. This gives a visual comparison on the count of true/false predictions from the dt model against the expected values(y_test)

[84]:
```
# histogram of predictions Descision Tree
plt.hist([dt_predicted,y_test], bins=2, label=['Model Predictions','True␣
↪Values'])
plt.title('Histogram of predictions Descision Tree')
plt.xlabel('Predictions')
plt.ylabel('Frequency')
plt.legend(loc='upper right')
plt.show()
```



Computing model accuracy scores using SKlearns metrics modules. First create a confusion matrix class object, then pass to it the test and predicted values from our model. The results of this matrix are passed to seaborn heatmap for output

[85]:
```
# calculte confusion matrix
dt_cm = confusion_matrix(y_test, dt_predicted)

# sns heatmap for the confusion matrix
sns.heatmap(dt_cm.T, square= True, annot=True, fmt='d',cbar=False)

plt.title('Confusion Matrix Decision Tree')
plt.xlabel('True Class')
plt.ylabel('Predicted Class')
plt.show()
```



Computing other metric scores for the model

[86]:
```
tp = dt_cm[0][0]
fp = dt_cm[0][1]
fn = dt_cm[1][0]
tn = dt_cm[1][1]

# print tp,fp,fn,tn
print('==========================================')
print('tp:', tp,'fp: ', fp, 'fn:', fn, 'tn:', tn)

# calculate accuracy  (tp + tn) / (p + n)
accuracy = accuracy_score(y_test, dt_predicted)
print('==========================================')
print ('Accuracy: ', accuracy)

# precision tp / (tp + fp)
precision = tp/(tp+fp)
print('==========================================')
print('Precision: ', precision)

# recall: tp / (tp + fn)
recall = tp / (tp+fn)
print('==========================================')
```

```
print('Recall: ', recall)

# f1: 2*Precision * Recall / (Prescion + Recall)
f1 = 2*precision * recall / (precision + recall)
print('==========================================')
print('F1 score: ', f1)
```

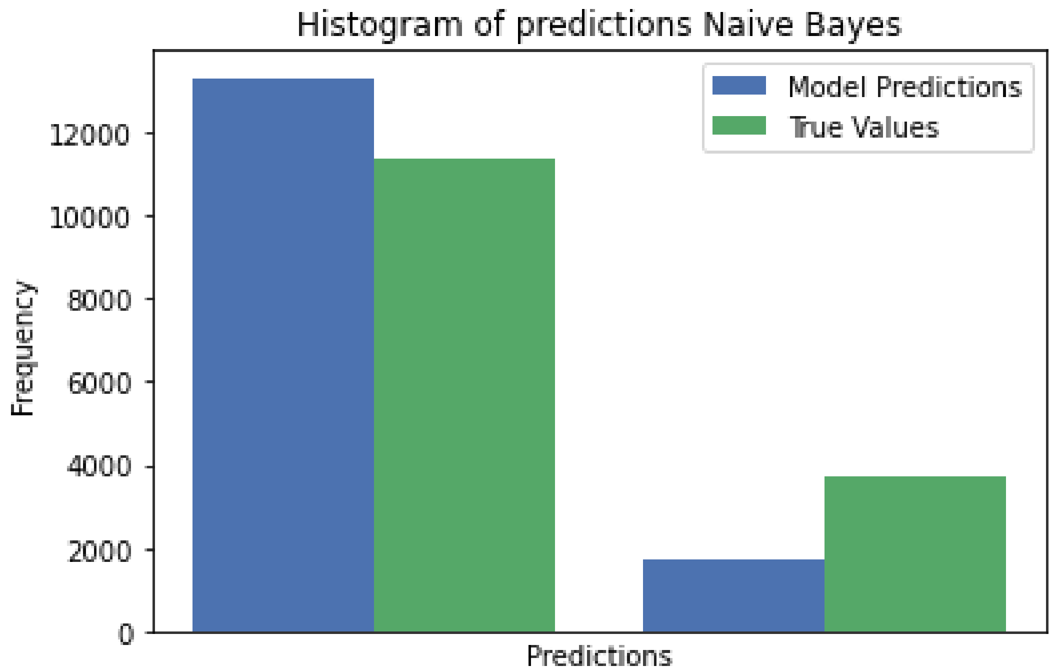
```
==========================================
tp: 9819 fp:  1541 fn: 1410 tn: 2290
==========================================
Accuracy:  0.8040504648074369
==========================================
Precision:  0.8643485915492958
==========================================
Recall:  0.8744322735773443
==========================================
F1 score:  0.8693611935012616
```

Naive Bayes Class.

[ ]:
```
############################
# Naive Bayesian Classifier #
############################



from sklearn.naive_bayes import GaussianNB

print("\nNaive Bayesian Classifier")

# make naive bayes Classifier
nb = GaussianNB()

# fit train data
nb.fit(X_train, y_train)
```

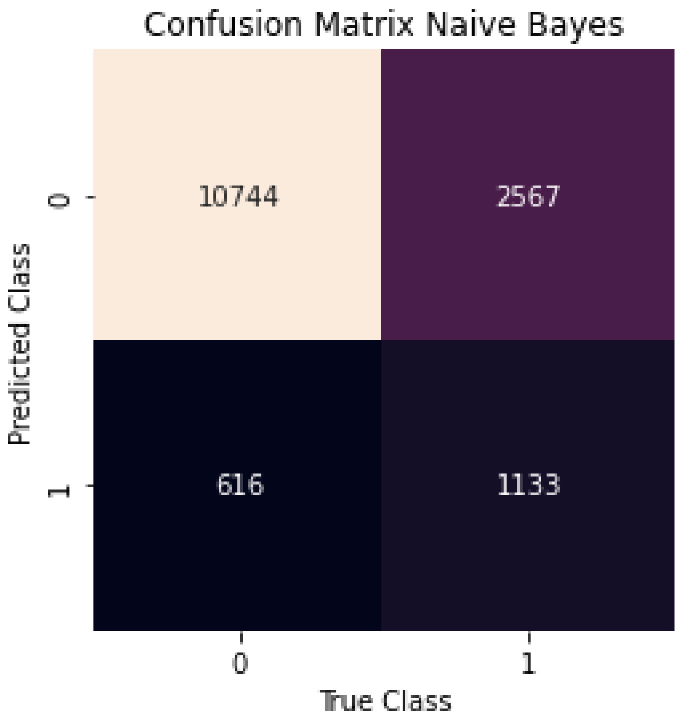
[88]:
```
# calculate prediction
nb_predicted = nb.predict(X_test);
#print( predicted)
```

[89]:
```
# histogram of predictions Naive Bayes
plt.hist([nb_predicted,y_test], bins=2, label=['Model Predictions','True␣
↪Values'])
plt.title('Histogram of predictions Naive Bayes')
plt.xlabel('Predictions')
plt.ylabel('Frequency')
plt.legend(loc='upper right')
plt.show()
```



[90]:

```
# calculte confusion matrix
nb_cm = confusion_matrix(y_test, nb_predicted)

# sns heatmap for the confusion matrix
sns.heatmap(nb_cm.T, square= True, annot=True, fmt='d',cbar=False)

plt.title('Confusion Matrix Naive Bayes')
plt.xlabel('True Class')
plt.ylabel('Predicted Class')
plt.show()
```



[91]:
```
tp = nb_cm[0][0]
fp = nb_cm[0][1]
fn = nb_cm[1][0]
tn = nb_cm[1][1]

print('==========================================')
print('tp:', tp,'fp: ', fp, 'fn:', fn, 'tn:', tn)

# show accuracy
accuracy = accuracy_score(y_test, nb_predicted)
print('==========================================')
print ('Accuracy: ', accuracy)

# precision tp / (tp + fp)
precision = tp/(tp+fp)
print('==========================================')
print('Precision: ', precision)

# recall: tp / (tp + fn)
recall = tp / (tp+fn)
print('==========================================')
print('Recall: ', recall)
```

```
# f1: 2*Precision * Recall / (Prescion + Recall)
f1 = 2*precision * recall / (precision + recall)
print('==========================================')
print('F1 score: ', f1)
```

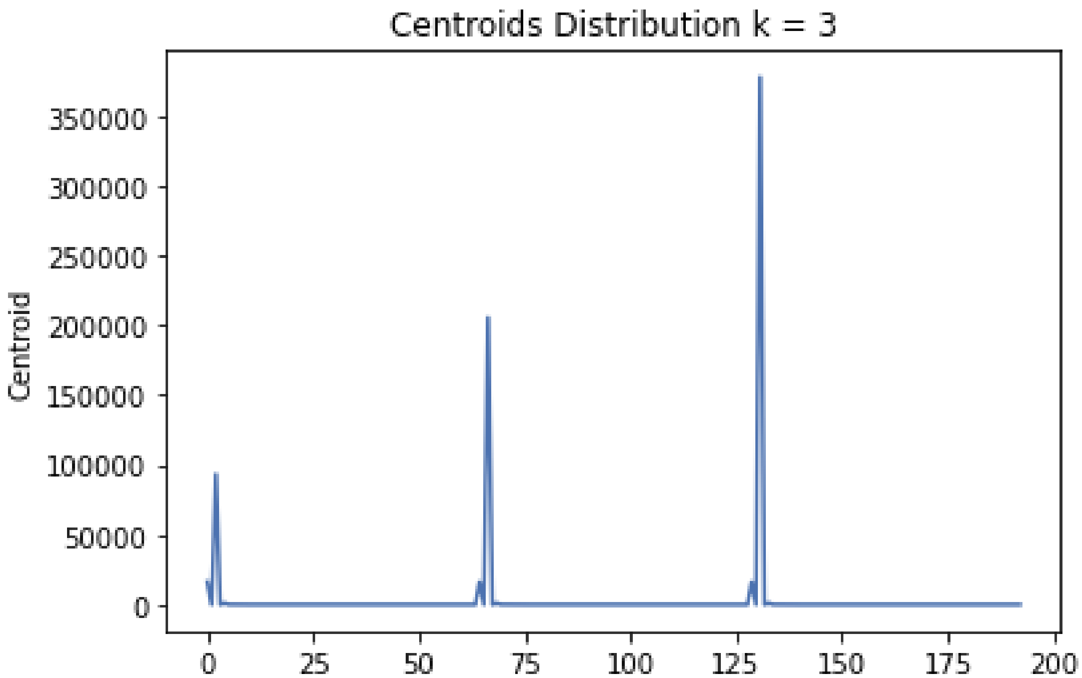
```
==========================================
tp: 10744 fp:  616 fn: 2567 tn: 1133
==========================================
Accuracy:  0.7886454183266932
==========================================
Precision:  0.9457746478873239
==========================================
Recall:  0.8071519795657727
==========================================
F1 score:  0.8709821247618661
```

K-Means Class.

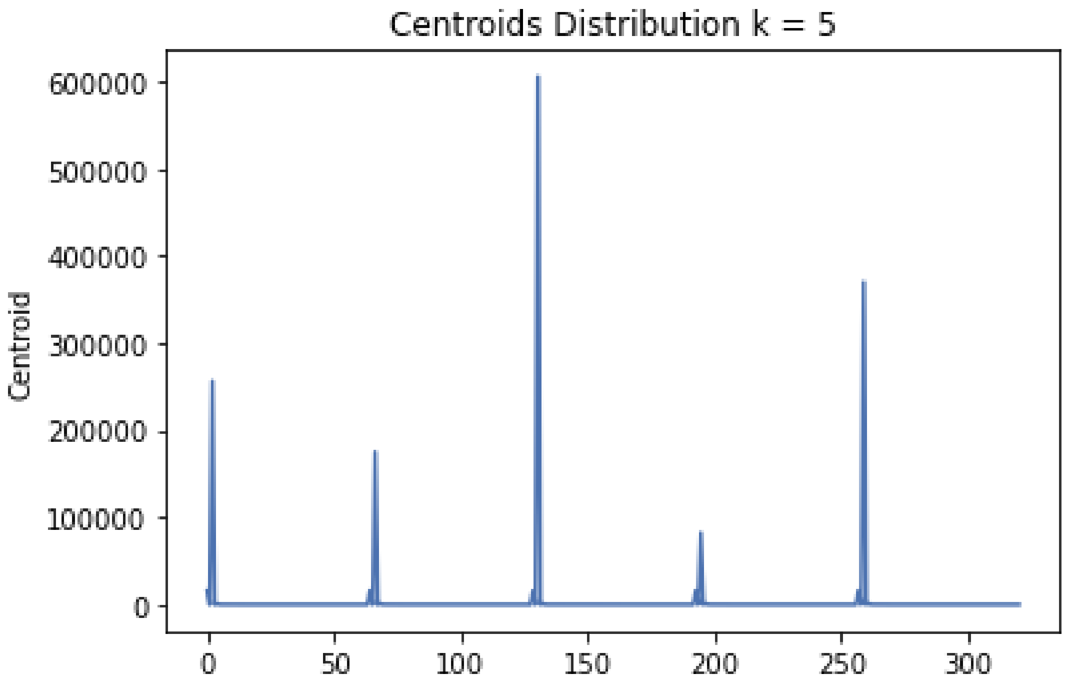
[93]:
```
########################
# K - Means Classifier #
########################


from sklearn.cluster import KMeans


print("\nK-Means Classifier")

# fit train data for k = 3,5,10
klist = [3,5,10]


# for each k
for k in klist:
    print('==========================================================')
    print("k = ",k)

    # make KMeans Classifier
    kmeans = KMeans(n_clusters=k)

    # fit train data
    kmeans.fit(X_train, y_train)

    # centroid cluster centers
    centroids = kmeans.cluster_centers_

    y = centroids.reshape(-1,1)
```

```
    x_ = y.shape[0]
    x = np.linspace(0, x_, num=x_)

    plt.plot(x, y)
    plt.title(f'Centroids Distribution k = {k}')
    plt.ylabel('Centroid')
    plt.show()
```

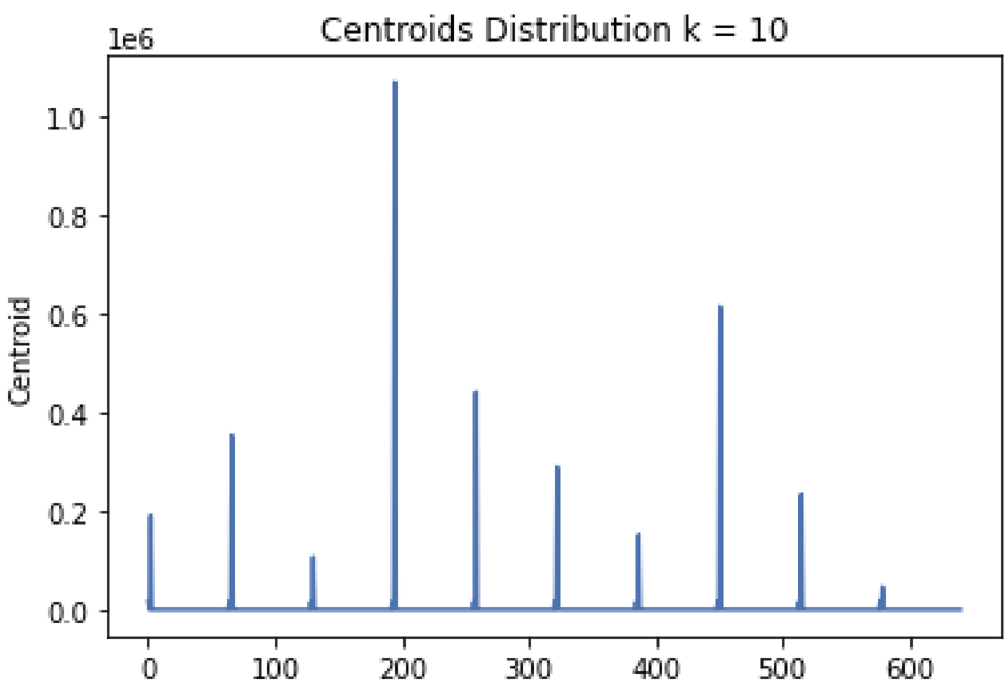
```
K-Means Classifier
==========================================================
k =  3
```




```
==========================================================
k =  5
```


Centroids Distribution k = 5
Centroid
600000
500000
400000
300000
200000
100000
0
0
50
100
150
200
250
300

```
============================================================
k =  10
```

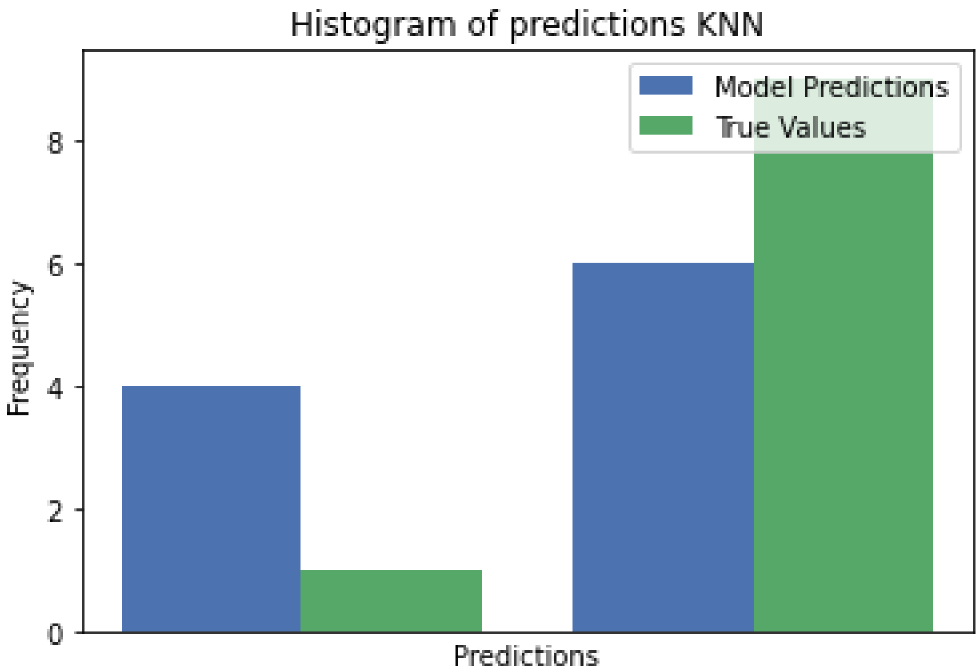

Centroids Distribution k = 10
1e6
Centroid
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500
600

KNN Classifiers

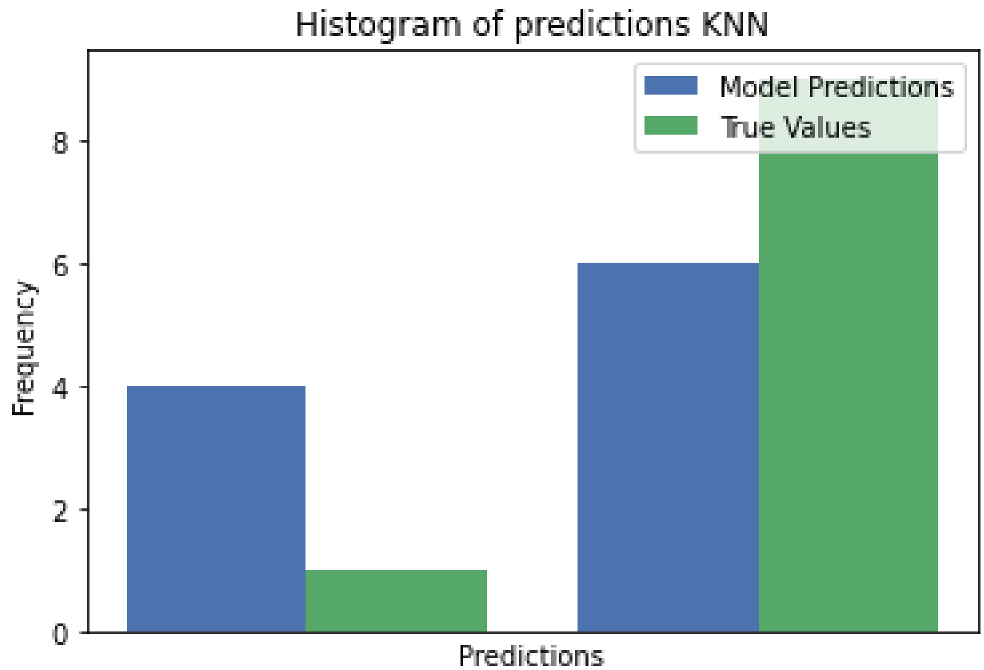
[94]:
```
##################
# KNN Classifier #
##################


from sklearn.neighbors import KNeighborsClassifier

print("\nKnn Classifier")


# last 10 records of test data  X,y
X_test = test_data[test_data.columns.drop('salary')][-10:]
y_test = test_data['salary'][-10:]


# fit train data for k = 3,5,10
klist = [3,5,10]

# for each k
for k in klist:
    print('===========================================================')
    print("k = ",k)

    # make KMeans Classifier
    knn = KNeighborsClassifier(n_neighbors=k)

    # fit train data
    knn.fit(X_train, y_train)

    # show test case prediction
    knn_predicted = dt.predict(X_test);
    #print( predicted)

    #plotting predictions
    plt.hist([knn_predicted,y_test], bins=2, label=['Model Predictions','True␣
↪Values'])
    plt.title('Histogram of predictions KNN')
    plt.xlabel('Predictions')
    plt.ylabel('Frequency')
    plt.legend(loc='upper right')
    plt.show()

    # calculte confusion matrix
    cm = confusion_matrix(y_test, knn_predicted)
    #print(cm)
    tp = cm[0][0]
```

```python
    fp = cm[0][1]
    fn = cm[1][0]
    tn = cm[1][1]

    # print tp,fp,fn,tn
    print('-----------------------------------------')
    print('tp:', tp,'fp: ', fp, 'fn:', fn, 'tn:', tn)

    # calculate accuracy  (tp + tn) / (p + n)
    accuracy = accuracy_score(y_test, knn_predicted)
    print('-----------------------------------------')
    print ('Accuracy: ', accuracy)

    # precision tp / (tp + fp)
    precision = tp/(tp+fp)
    print('-----------------------------------------')
    print('Precision: ', precision)

    # recall: tp / (tp + fn)
    recall = tp / (tp+fn)
    print('-----------------------------------------')
    print('Recall: ', recall)

    # f1: 2*Precision * Recall / (Prescion + Recall)
    f1 = 2*precision * recall / (precision + recall)
    print('-----------------------------------------')
    print('F1 score: ', f1)
```

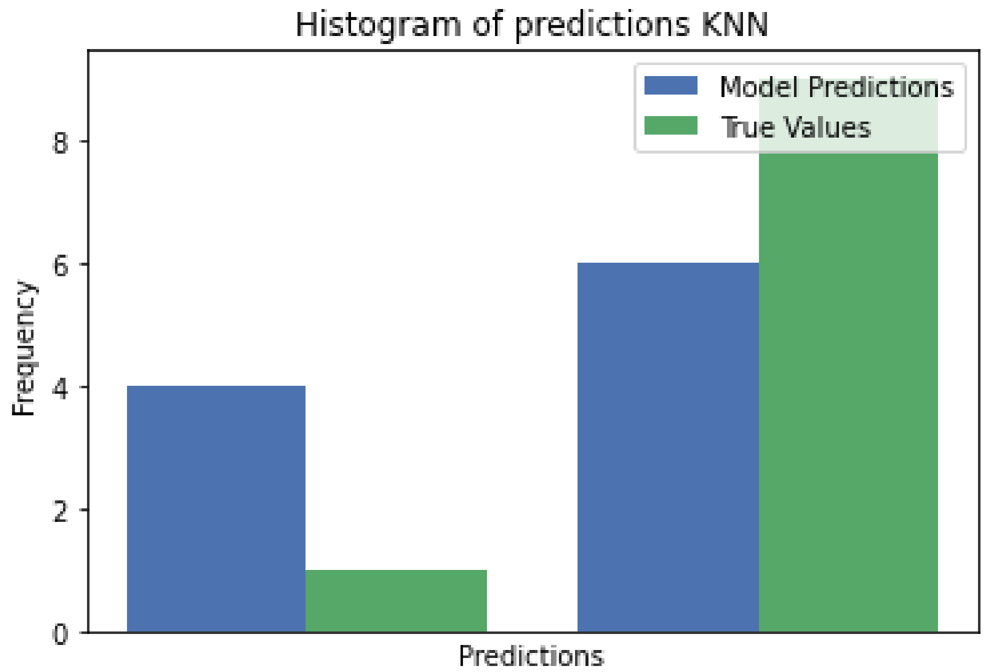
```
Knn Classifier
==========================================================
k =  3
```




```
-----------------------------------------
tp: 6 fp:  3 fn: 0 tn: 1
-----------------------------------------
Accuracy:  0.7
-----------------------------------------
Precision:  0.6666666666666666
-----------------------------------------
Recall:  1.0
-----------------------------------------
F1 score:  0.8
==========================================================
k =  5
```




```
-----------------------------------------
tp: 6 fp:  3 fn: 0 tn: 1
-----------------------------------------
Accuracy:  0.7
-----------------------------------------
Precision:  0.6666666666666666
-----------------------------------------
Recall:  1.0
-----------------------------------------
F1 score:  0.8
==========================================================
k =  10
```




```
-----------------------------------------
tp: 6 fp:  3 fn: 0 tn: 1
-----------------------------------------
Accuracy:  0.7
-----------------------------------------
Precision:  0.6666666666666666
-----------------------------------------
Recall:  1.0
-----------------------------------------
F1 score:  0.8
```

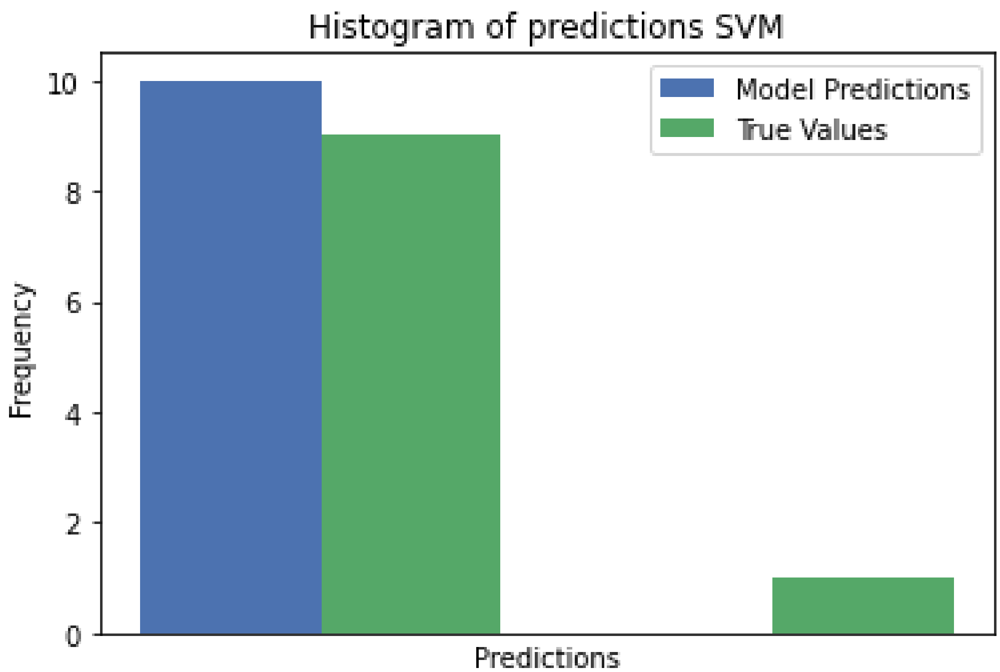
SVM CLass.

[ ]:
```
##################
# SVM Classifier #
##################


from sklearn import svm

print("\nSVM Classifier")

# make SVM Classifier
sv = svm.SVC()

# fit train data
```

```
sv.fit(X_train, y_train)
```

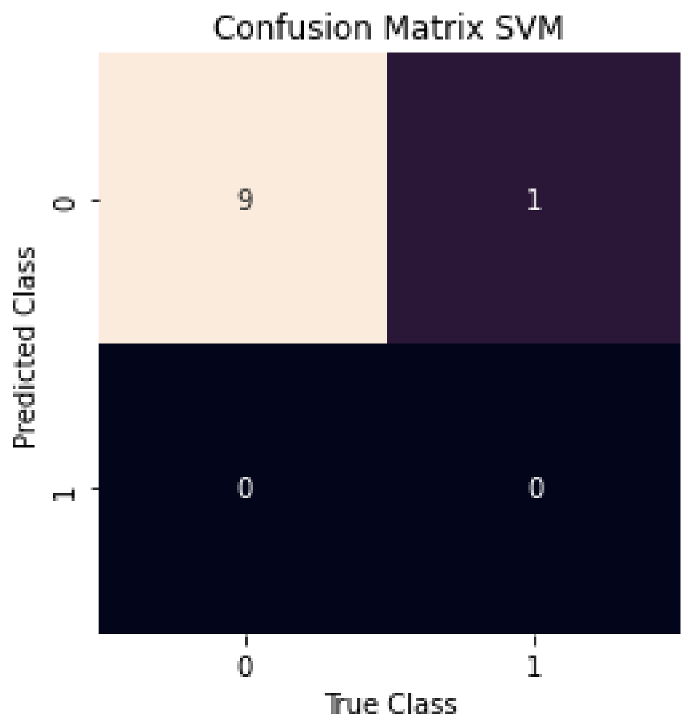
[96]:
```
# predict test case
sv_predicted = sv.predict(X_test);
#print( predicted)
```

[97]:
```
# histogram of predicted probabilities SVM
plt.hist([sv_predicted,y_test], bins=2, label=['Model Predictions','True␣
 ↪Values'])
plt.title('Histogram of predictions SVM')
plt.xlabel('Predictions')
plt.ylabel('Frequency')
plt.legend(loc='upper right')
plt.show()
```



[101]:
```
# calculte confusion matrix
sv_cm = confusion_matrix(y_test, sv_predicted)

# sns heatmap for the confusion matrix
sns.heatmap(sv_cm.T, square= True, annot=True, fmt='d',cbar=False)

plt.title('Confusion Matrix SVM')
plt.xlabel('True Class')
plt.ylabel('Predicted Class')
```

```
plt.show()
```



[102]:
```
tp = sv_cm[0][0]
fp = sv_cm[0][1]
fn = sv_cm[1][0]
tn = sv_cm[1][1]

# print tp,fp,fn,tn
print('==========================================')
print('tp:', tp,'fp: ', fp, 'fn:', fn, 'tn:', tn)

# show accuracy  (tp + tn) / (p + n)
accuracy = accuracy_score(y_test, sv_predicted)
print('==========================================')
print ('Accuracy: ', accuracy)

# precision tp / (tp + fp)
precision = tp/(tp+fp)
print('==========================================')
print('Precision: ', precision)

# recall: tp / (tp + fn)
recall = tp / (tp+fn)
```

```
print('=========================================')
print('Recall: ', recall)

# f1: 2*Precision * Recall / (Prescion + Recall)
f1 = 2*precision * recall / (precision + recall)
print('=========================================')
print('F1 score: ', f1)
```

```
=========================================
tp: 9 fp:  0 fn: 1 tn: 0
=========================================
Accuracy:  0.9
=========================================
Precision:  1.0
=========================================
Recall:  0.9
=========================================
F1 score:  0.9473684210526316
```

Neural Net CLassifier

[104]:
```
##################
# MLP Classifier #
##################



from sklearn.neural_network import MLPClassifier

print("\nMLP Classifier")



# make MLP Classifier
mlp = MLPClassifier(random_state=1, max_iter=300)

# fit train data
mlp.fit(X_train, y_train)
```

[104]:
```
MLPClassifier(max_iter=300, random_state=1)
```

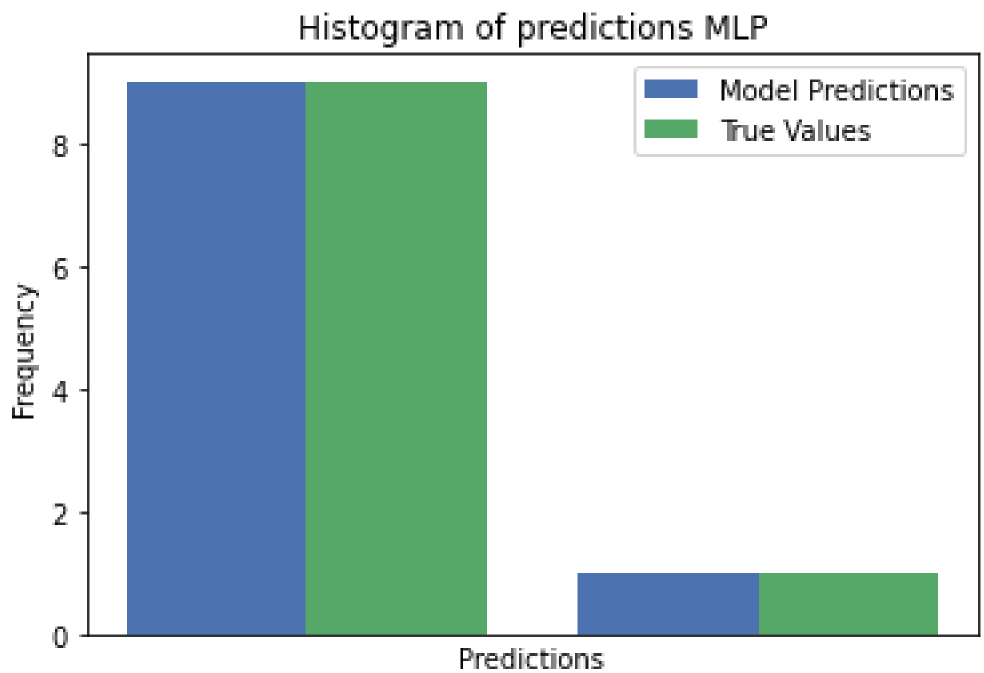
[105]:
```
# print probailities
probabilities = mlp.predict_proba(X_test)
print("probabilities")
print(probabilities)
```

```
probabilities
[[1.00000000e+000 1.10482062e-085]
 [9.99999960e-001 3.95065516e-008]
 [1.00000000e+000 1.97640018e-141]
```

```
[1.00000000e+000 5.87673593e-146]
[1.00000000e+000 1.88467617e-116]
[1.00000000e+000 6.18129057e-083]
[1.00000000e+000 7.99328556e-071]
[1.00000000e+000 2.49689439e-096]
[0.00000000e+000 1.00000000e+000]
[1.00000000e+000 1.42692344e-020]]
```

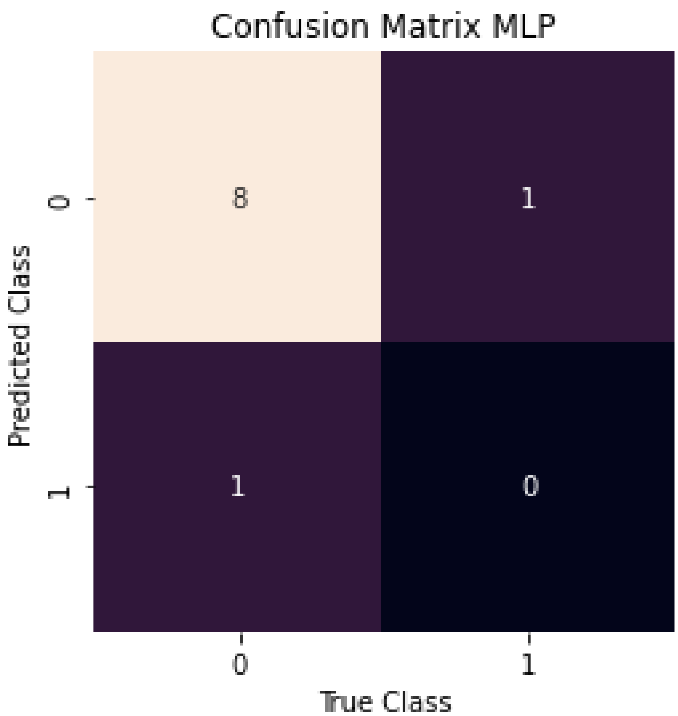
[106]:
```
# predict
mlp_predicted = mlp.predict(X_test)
#print(predicted)
```

[107]:
```
# histogram of predicted probabilities MLP
plt.hist([mlp_predicted,y_test], bins=2, label=['Model Predictions','True␣
↪Values'])
plt.title('Histogram of predictions MLP')
plt.xlabel('Predictions')
plt.ylabel('Frequency')
plt.legend(loc='upper right')
plt.show()
```



[108]:
```
# calculte confusion matrix
mlp_cm = confusion_matrix(y_test, mlp_predicted)

# sns heatmap for the confusion matrix
```

```
sns.heatmap(mlp_cm.T, square= True, annot=True, fmt='d',cbar=False)

plt.title('Confusion Matrix MLP')
plt.xlabel('True Class')
plt.ylabel('Predicted Class')
plt.show()
```



[109]:

```
tp = mlp_cm[0][0]
fp = mlp_cm[0][1]
fn = mlp_cm[1][0]
tn = mlp_cm[1][1]

# print tp,fp,fn,tn
print('========================================')
print('tp:', tp,'fp: ', fp, 'fn:', fn, 'tn:', tn)

# calculate accuracy  (tp + tn) / (p + n)
accuracy = accuracy_score(y_test, mlp_predicted)
print('========================================')
print ('Accuracy: ', accuracy)

# precision tp / (tp + fp)
precision = tp/(tp+fp)
```

```python
print('========================================')
print('Precision: ', precision)

# recall: tp / (tp + fn)
recall = tp / (tp+fn)
print('========================================')
print('Recall: ', recall)

# f1: 2*Precision * Recall / (Prescion + Recall)
f1 = 2*precision * recall / (precision + recall)
print('========================================')
print('F1 score: ', f1)
```

```
========================================
tp: 8 fp:  1 fn: 1 tn: 0
========================================
Accuracy:  0.8
========================================
Precision:  0.8888888888888888
========================================
Recall:  0.8888888888888888
========================================
F1 score:  0.8888888888888888
```

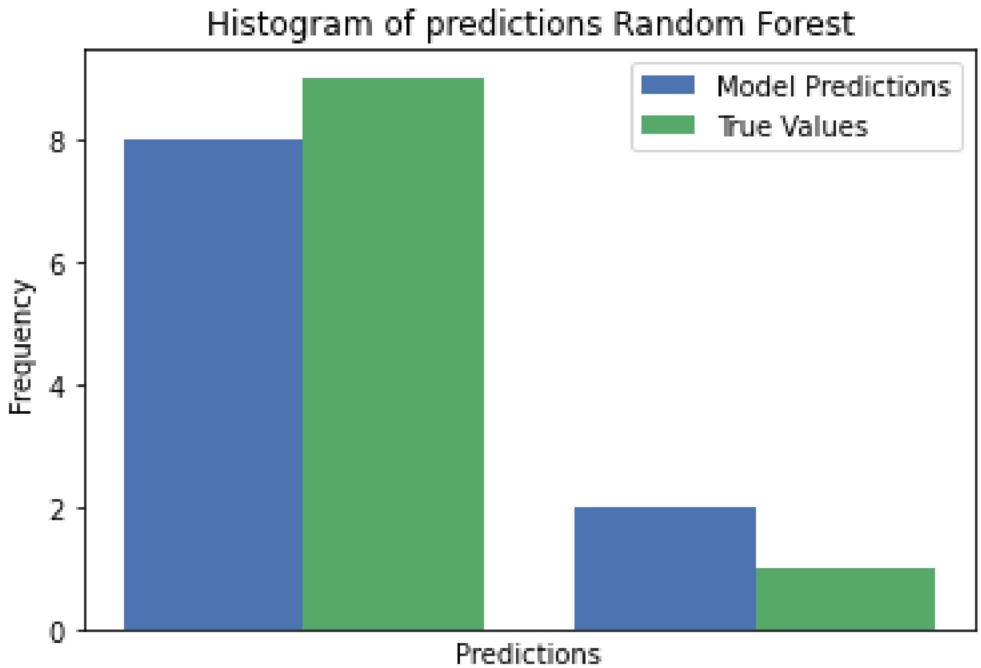
Random Forest

[ ]:
```python
############################
# Random Forest Classifier #
############################


from sklearn.ensemble import RandomForestClassifier

print("\nRandom Forest Classifier")

# make SVM Classifier
rf = RandomForestClassifier(random_state=0)

# fit train data
rf.fit(X_train, y_train)
```

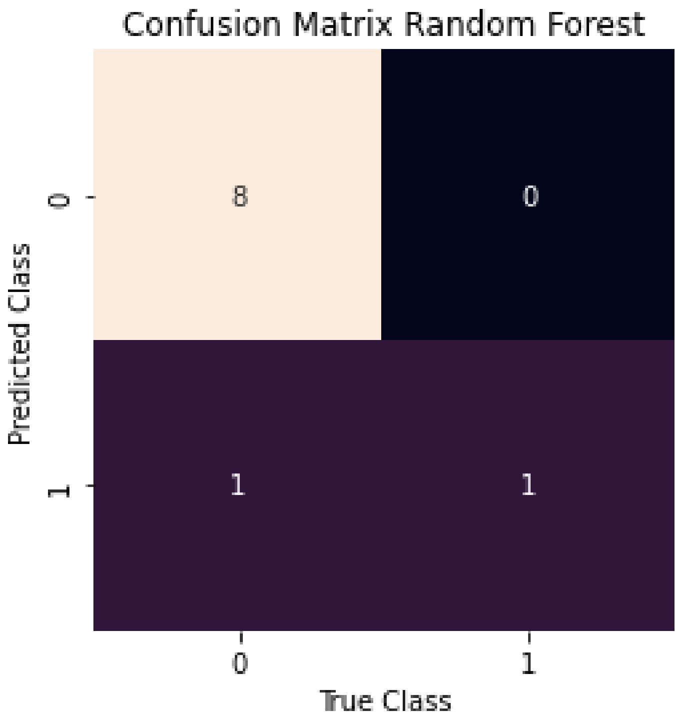
[111]:
```python
# predict test case
rf_predicted = rf.predict(X_test);
#print( predicted)
```

[112]:
```python
# histogram of predictions Random Forest
plt.hist([rf_predicted,y_test], bins=2, label=['Model Predictions','True␣
↪Values'])
```

```
plt.title('Histogram of predictions Random Forest')
plt.xlabel('Predictions')
plt.ylabel('Frequency')
plt.legend(loc='upper right')
plt.show()
```



[113]:
```
# calculte confusion matrix
rf_cm = confusion_matrix(y_test, rf_predicted)

# sns heatmap for the confusion matrix
sns.heatmap(rf_cm.T, square= True, annot=True, fmt='d',cbar=False)

plt.title('Confusion Matrix Random Forest')
plt.xlabel('True Class')
plt.ylabel('Predicted Class')
plt.show()
```



[114]:
```
tp = rf_cm[0][0]
fp = rf_cm[0][1]
fn = rf_cm[1][0]
tn = rf_cm[1][1]

# print tp,fp,fn,tn
print('========================================')
print('tp:', tp,'fp: ', fp, 'fn:', fn, 'tn:', tn)

# show accuracy  (tp + tn) / (p + n)
rf_accuracy = accuracy_score(y_test, rf_predicted)
print('========================================')
print ('Accuracy: ', rf_accuracy)

# precision tp / (tp + fp)
precision = tp/(tp+fp)
print('========================================')
print('Precision: ', precision)

# recall: tp / (tp + fn)
recall = tp / (tp+fn)
print('========================================')
print('Recall: ', recall)
```

```
# f1: 2*Precision * Recall / (Prescion + Recall)
f1 = 2*precision * recall / (precision + recall)
print('========================================')
print('F1 score: ', f1)
```

```
========================================
tp: 8 fp:  1 fn: 0 tn: 1
========================================
Accuracy:  0.9
========================================
Precision:  0.8888888888888888
========================================
Recall:  1.0
========================================
F1 score:  0.9411764705882353
```

```
[ ]:
```