Machine Learning for Mental Health in Tech

CS-E4875 - Research Project Report

Introduction

Mental health disorders constitute a global public health challenge, affecting millions of individuals across the world, irrespective of age, gender, or socio-economic status. According to the World Health Organization (WHO), approximately 1 in 4 people will experience a mental health issue at some point in their lives, making it a prevalent and pressing concern. These disorders can have a profound impact on an individual's overall well-being, daily functioning, and overall quality of life. Early detection and intervention are critical in managing and treating mental health conditions effectively. In this context, the field of machine learning (ML) has emerged as a promising avenue for improving mental health prediction and support.

In recent years, ML techniques have gained prominence in the healthcare sector, offering innovative approaches for understanding, diagnosing, and predicting various medical conditions. Within the realm of mental health, ML has the potential to revolutionize the way we approach the identification, assessment, and treatment of mental health disorders.

The objective of this research project is to explore the application of machine learning in the context of mental health prediction. By leveraging advanced ML algorithms and techniques, we aim to develop predictive models that can assist healthcare professionals and individuals in identifying and managing mental health issues early on. This project will delve into various aspects of machine learning, including data collection, preprocessing, feature extraction, model development, and evaluation, all tailored to the unique challenges and nuances of mental health prediction.

Problem Formulation

The central focus of this research project is to address the critical question of whether machine learning algorithms can predict the necessity of treatment for patients with mental illness based on the data available in a given dataset. The problem formulation can be structured as follows:

The core problem addressed in this research is the prediction of the need for treatment for individuals with mental health conditions. The primary question to be answered is whether machine learning models can effectively determine, based on the values extracted from a dataset, whether a patient should undergo treatment for their mental illness. This predictive task encompasses a wide range of mental health conditions, such as depression, anxiety, bipolar disorder, and others.

The research project relies on the availability of a comprehensive and well-structured dataset containing relevant features and data points associated with individuals' mental health. These features may encompass demographic information, clinical assessments, medical history, symptom severity, treatment history, and potentially other variables that are indicative of the patient's mental health status and their need for treatment.

Methods

In this research project three different machine learning algorithms are used. The purpose is to first test simpler and then more complex models and see their differences.

The first algorithm implemented and tested is logistic regression which is quite simple model and is also suitable for small datasets. The second algorithm implemented and tested is the K nearest neighbor classifier and it also works for small data sets. The third algorithm implemented and tested is a neural network which usually require larger datasets but I decided to implement it to see how it performs in this project setting.

The loss function used in this project varies depending on the model used. The logistic regression uses the log loss function. The K nearest neighbor classifier doesn’t have a loss function in a sense. The neural network uses softmax cross entropy.

The training set and test set are constructed with random split. The training set is 70% of the dataset and the test set is 30% of the dataset. This split choice is based on general split suggestions and good practices.

Data
The data is obtained from kaggle.com and it contains 1259 data points. The label in the data is the need for treatment and the features are listed below. The dataset is from a 2014 survey that measures frequency of mental health disorders in the tech workplace.

Features

The data contains the following features age, gender, country, timestamp, state, self employed, family history, treatment, work infere, no emplyees, remote work, tech company, benefits, care options, wellness program, seek help, anonymity, leave, mental health consequence, phys health consequence, coworkers, supervisor, mental health interview, phys health interview, mental vs physical, obs consequence, comments. For simplicity only the subset of features are selected for this project and they can be seen in the image 1: correlation matrix below. The features were chosen based on their predicted importance.

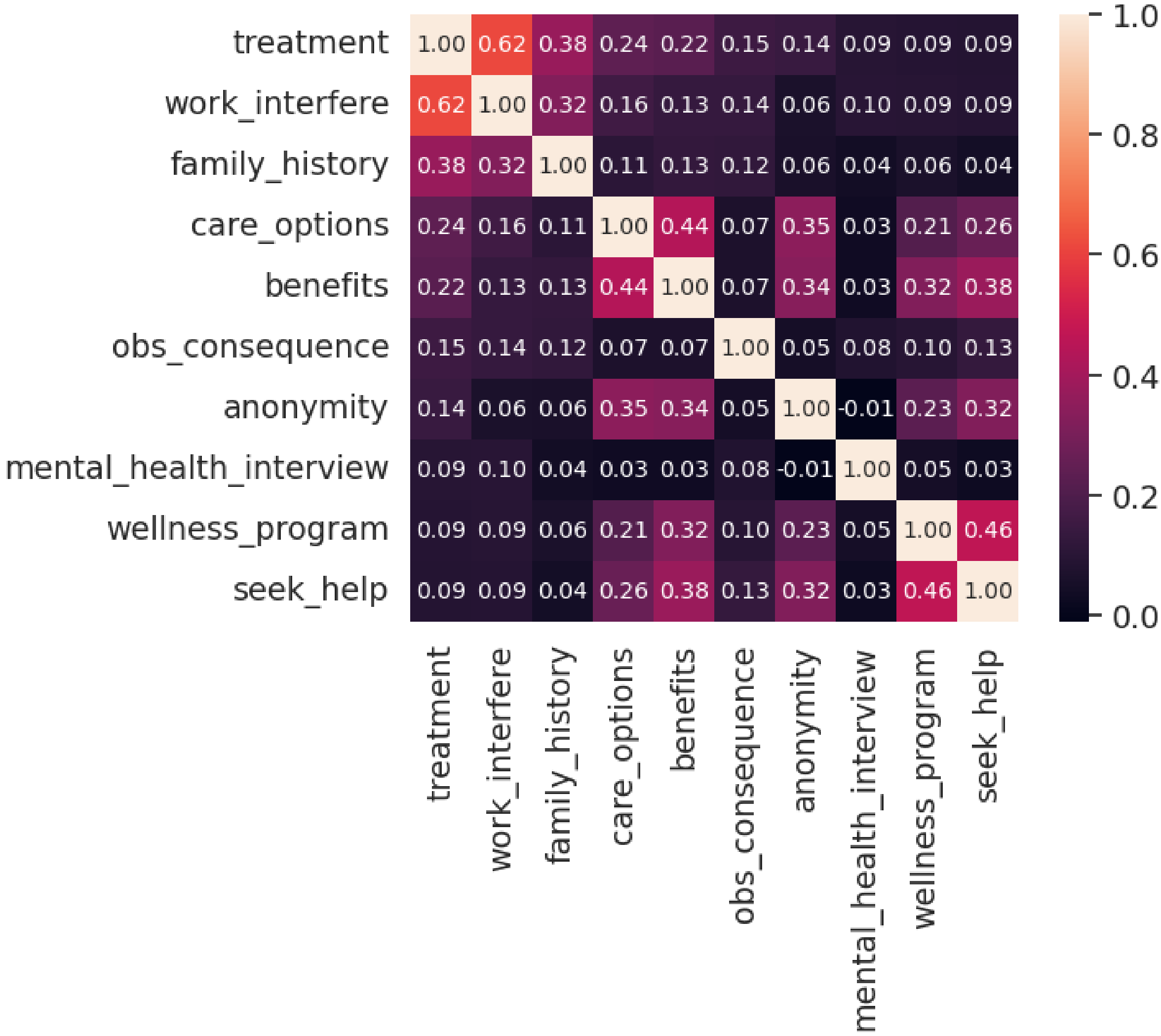


Image 1: Correlation Matrix

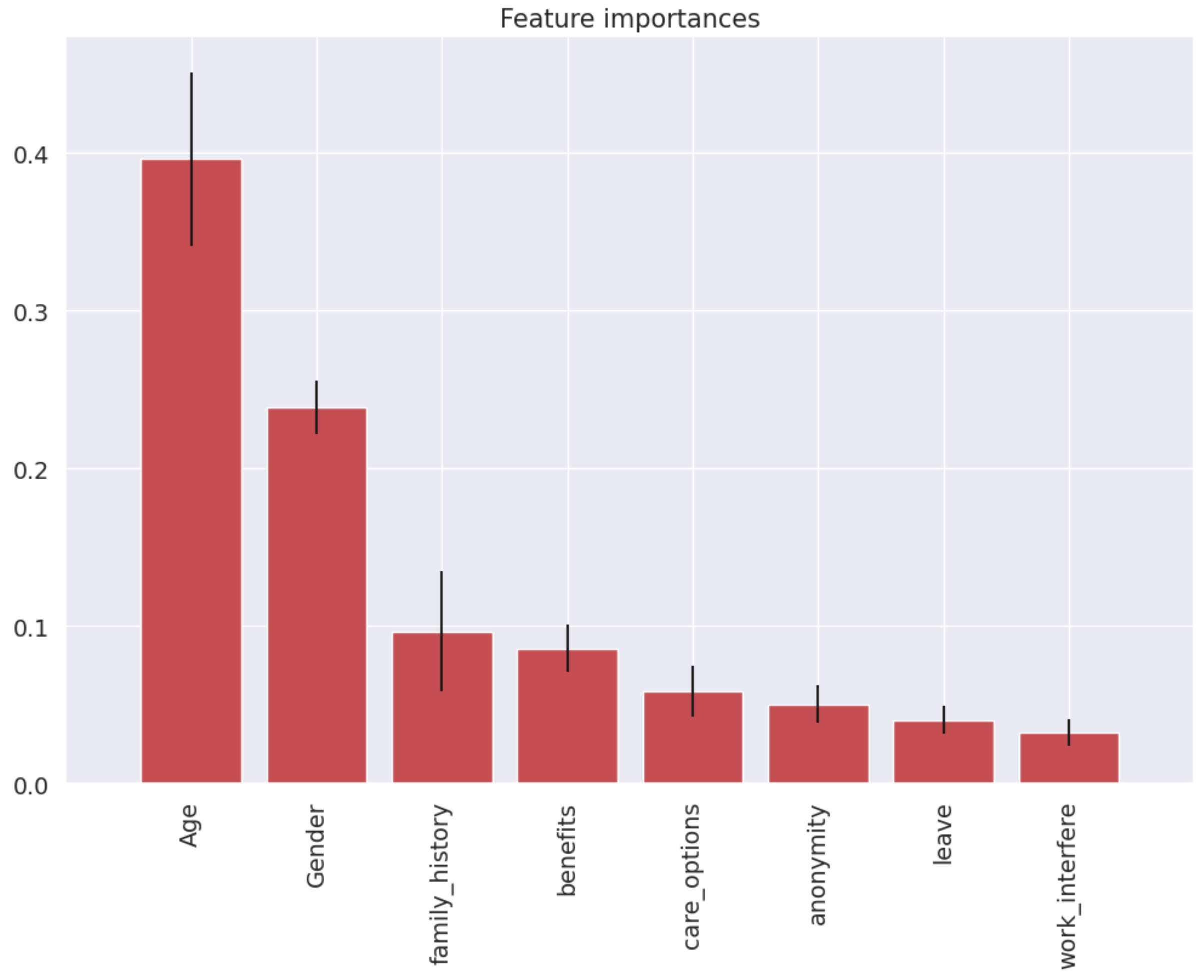


Image 2: Feature Importances

Results

The Results section of this research project provides a detailed analysis of the outcomes and findings obtained through the application of machine learning models to predict the necessity of treatment for individuals with mental illness. This section presents the key results, performance metrics, and insights derived from the predictive models.

All the results seem to be very close to each other and to perform equally well. The deep neural network model performs best, achieving test accuracy of 0.81, the second best model is knn with 0.7989 accuracy and the third best model is log regression with 0.7963.

The training accuracies seem also to be very close to each other. The neural network model achieved accuracy of 0.84, knn 0.82 and log regression 0.81.

Logistic regression:

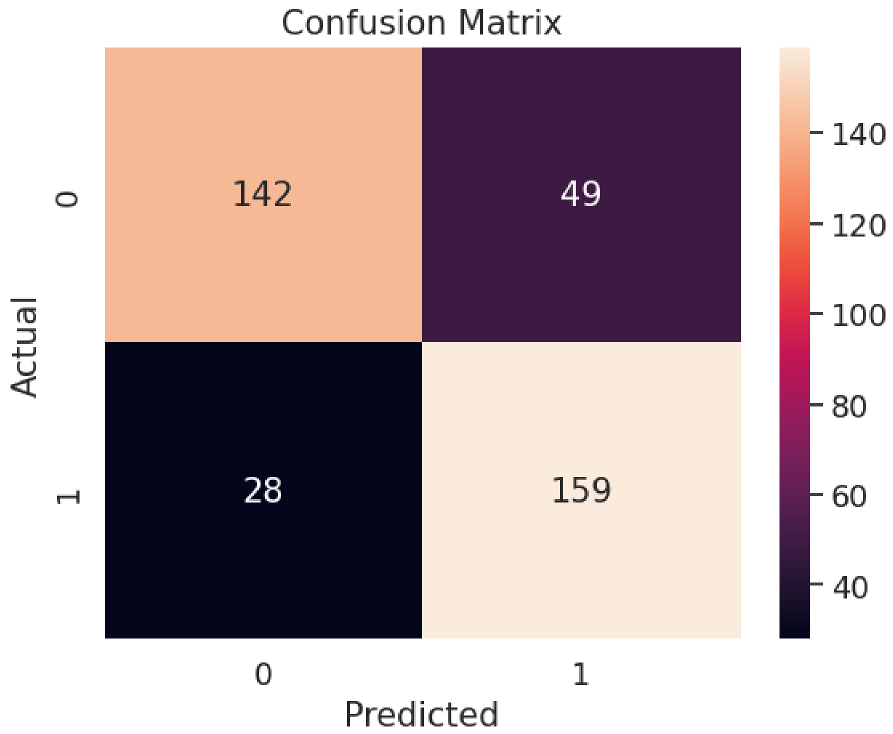


Image 3: Confusion Matrix for logistic regression

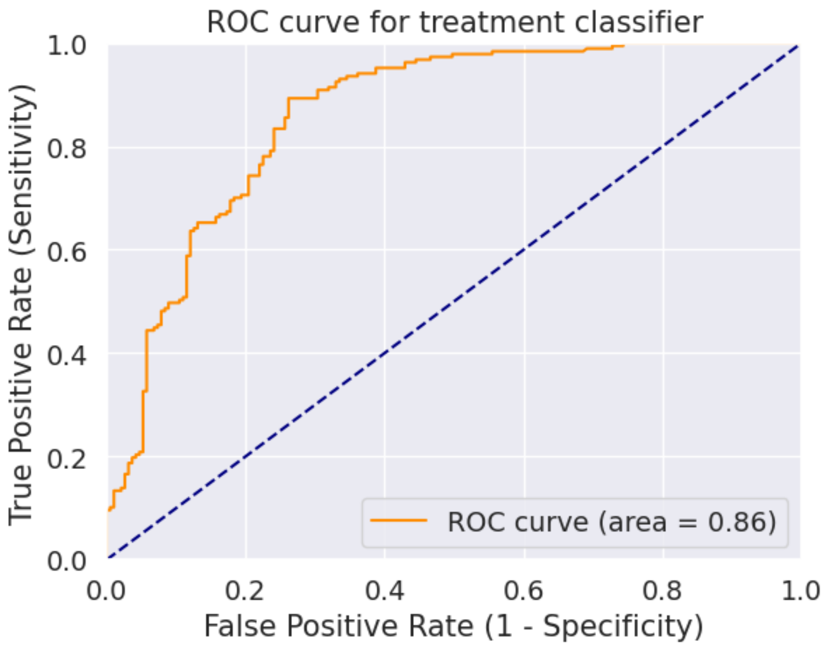


Image 4: ROC curve for logistic regression

K nearest neighbor:

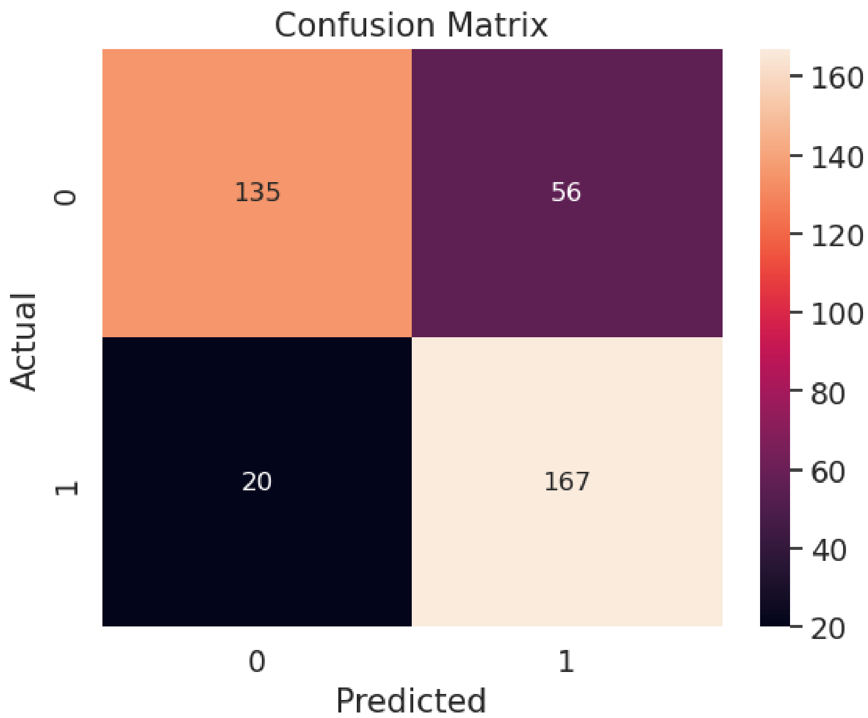


Image 5: Confusion matrix for K nearest neighbor classifier

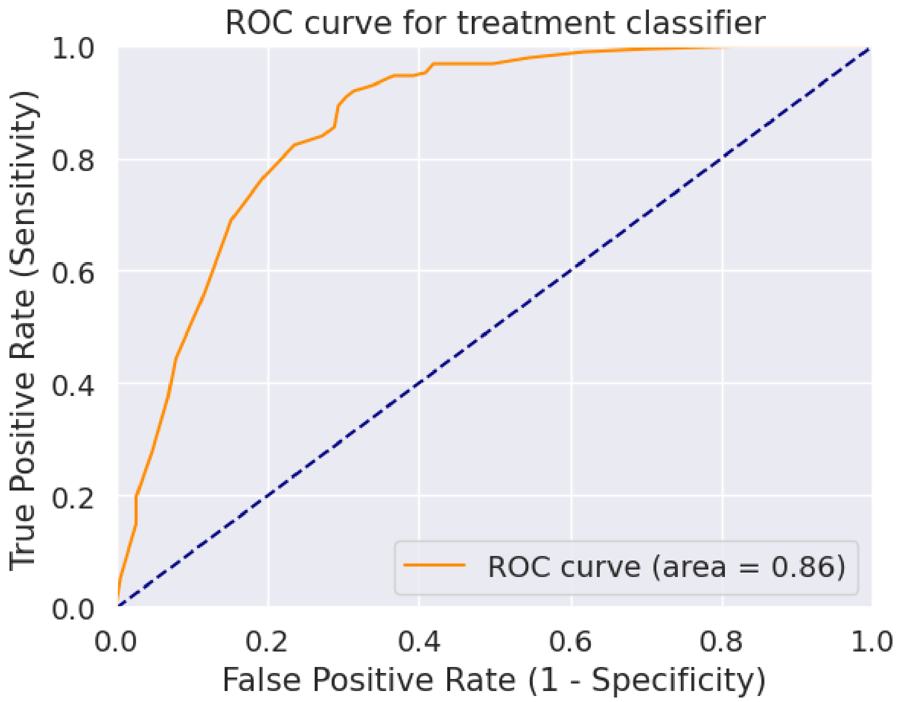


Image 6: ROC curve for K nearest neighbor classifier

Model Performance

Several machine learning models were trained and evaluated for their ability to predict whether patients with mental health conditions should undergo treatment.

The results obtained in this research project hold significant implications for the field of mental health and the use of machine learning in healthcare. The accuracy and performance metrics achieved by the predictive models suggest that machine learning can play a valuable role in assisting healthcare professionals in making informed decisions regarding the necessity of treatment for patients with mental health conditions. The identified features of importance offer insights into the factors that heavily influence these predictions.

However, it is essential to recognize the limitations of the research, such as the reliance on the quality and comprehensiveness of the dataset and the need for ongoing ethical considerations in the deployment of predictive models in healthcare.

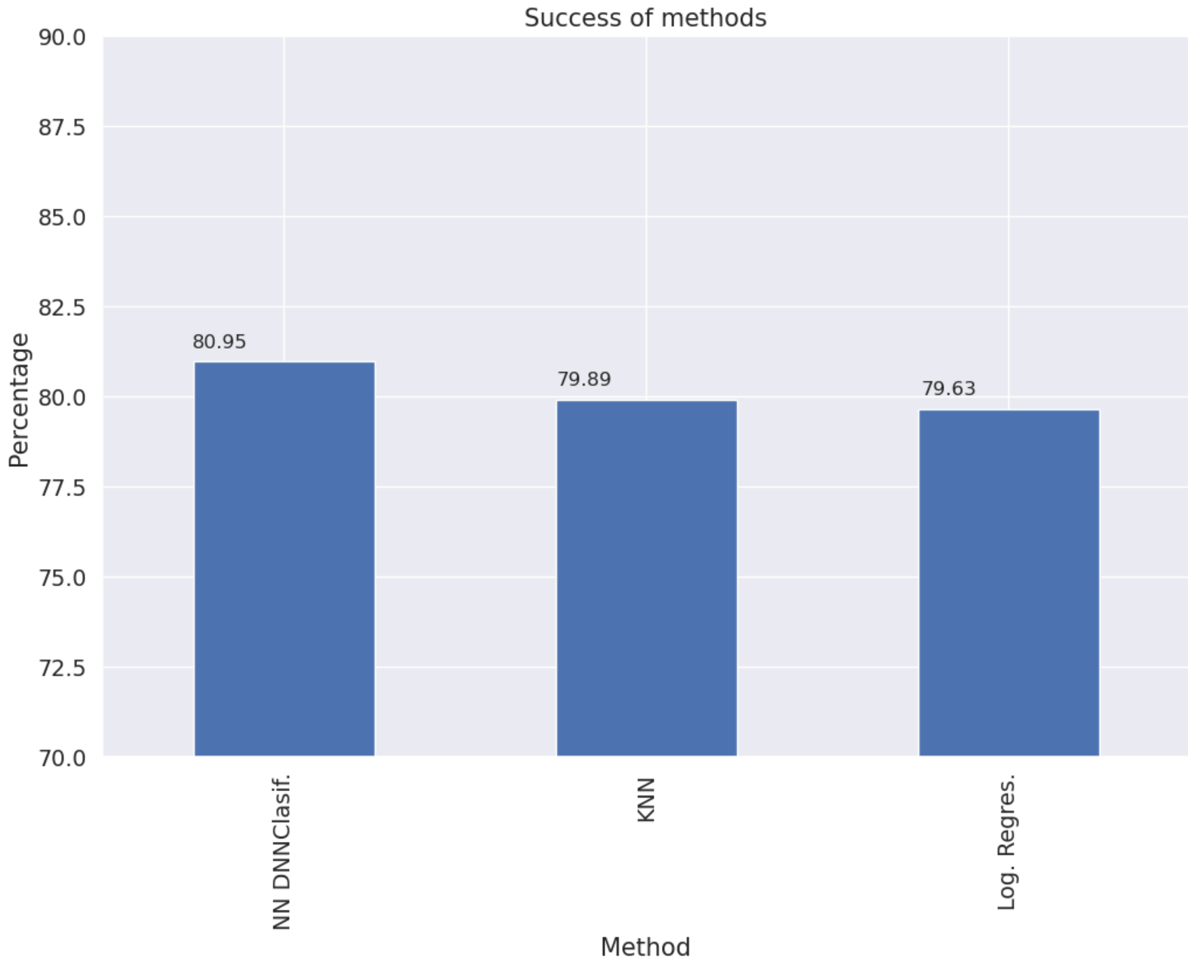


Image 7: Results of different methods

Conclusion

In this project I have implemented various different machine learning algorithms and tested their capability in predicting the need for mental health treatment. As the results indicate it is possible to predict the need for treatment with around 80% accuracy. This result seems relatively good taking into account the complexity of the whole mental health problem and its various different forms and the size of the relatively small dataset used.

The accuracy could be improved further in various ways. The most obvious way would be to gather more data by doing more surveys. This would quite likely improve the accuracy. Also having more data, more complex neural networks could be used, which in turn could improve the accuracy. In the space of machine learning also many other algorithms exist and those could also be tried and they could yield better accuracy.

References

https://www.kaggle.com/datasets/osmi/mental-health-in-tech-2016

Appendices

project

December 20, 2023

Library and data loading

[1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.stats import randint
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.datasets import make_classification
from sklearn.preprocessing import binarize, LabelEncoder, MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score, mean_squared_error,␣
 ↪precision_recall_curve
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from mlxtend.classifier import StackingClassifier

train_df = pd.read_csv('survey.csv')
```

Data Cleaning

[2]:
```
train_df = train_df.drop(['comments'], axis= 1)
train_df = train_df.drop(['state'], axis= 1)
train_df = train_df.drop(['Timestamp'], axis= 1)

train_df.isnull().sum().max()
train_df.head(5)
```

[2]:
```
   Age  Gender         Country self_employed family_history treatment  \
0   37  Female   United States           NaN             No       Yes
1   44       M   United States           NaN             No        No
2   32    Male          Canada           NaN             No        No
3   31    Male  United Kingdom           NaN            Yes       Yes
4   31    Male   United States           NaN             No        No

  work_interfere    no_employees remote_work tech_company  …   anonymity  \
0          Often            6-25          No          Yes  …         Yes
1         Rarely  More than 1000          No           No  …  Don't know
2         Rarely            6-25          No          Yes  …  Don't know
3          Often          26-100          No          Yes  …          No
4          Never         100-500         Yes          Yes  …  Don't know

                leave mental_health_consequence phys_health_consequence  \
0       Somewhat easy                        No                      No
1          Don't know                     Maybe                      No
2  Somewhat difficult                        No                      No
3  Somewhat difficult                       Yes                     Yes
4          Don't know                        No                      No

      coworkers supervisor mental_health_interview phys_health_interview  \
0  Some of them        Yes                      No                 Maybe
1            No         No                      No                    No
2           Yes        Yes                     Yes                   Yes
3  Some of them         No                   Maybe                 Maybe
4  Some of them        Yes                     Yes                   Yes

  mental_vs_physical obs_consequence
0                Yes              No
1         Don't know              No
2                 No              No
3                 No             Yes
4         Don't know              No

[5 rows x 24 columns]
```

[3]:
```python
defaultInt = 0
defaultString = 'NaN'
defaultFloat = 0.0

intFeatures = ['Age']
stringFeatures = ['Gender', 'Country', 'self_employed', 'family_history',␣
↪'treatment', 'work_interfere',
                 'no_employees', 'remote_work', 'tech_company', 'anonymity',␣
↪'leave', 'mental_health_consequence',
```

```python
                 'phys_health_consequence', 'coworkers', 'supervisor',␣
 ↪'mental_health_interview', 'phys_health_interview',
                 'mental_vs_physical', 'obs_consequence', 'benefits',␣
 ↪'care_options', 'wellness_program',
                 'seek_help']
floatFeatures = []

for feature in train_df:
    if feature in intFeatures:
        train_df[feature] = train_df[feature].fillna(defaultInt)
    elif feature in stringFeatures:
        train_df[feature] = train_df[feature].fillna(defaultString)
    elif feature in floatFeatures:
        train_df[feature] = train_df[feature].fillna(defaultFloat)
    else:
        print('Error: Feature %s not recognized.' % feature)
train_df.head(5)
```

[3]:

```
   Age  Gender         Country self_employed family_history treatment  \
0   37  Female   United States           NaN             No       Yes
1   44       M   United States           NaN             No        No
2   32    Male          Canada           NaN             No        No
3   31    Male  United Kingdom           NaN            Yes       Yes
4   31    Male   United States           NaN             No        No

  work_interfere    no_employees remote_work tech_company  …   anonymity  \
0          Often            6-25          No          Yes  …         Yes
1         Rarely  More than 1000          No           No  …  Don't know
2         Rarely            6-25          No          Yes  …  Don't know
3          Often          26-100          No          Yes  …          No
4          Never         100-500         Yes          Yes  …  Don't know

                 leave mental_health_consequence phys_health_consequence  \
0        Somewhat easy                        No                      No
1           Don't know                     Maybe                      No
2   Somewhat difficult                        No                      No
3   Somewhat difficult                       Yes                     Yes
4           Don't know                        No                      No

      coworkers supervisor mental_health_interview phys_health_interview  \
0  Some of them        Yes                      No                 Maybe
1            No         No                      No                    No
2           Yes        Yes                     Yes                   Yes
3  Some of them         No                   Maybe                 Maybe
4  Some of them        Yes                     Yes                   Yes

  mental_vs_physical obs_consequence
```

```
0                 Yes                No
1          Don't know                No
2                  No                No
3                  No               Yes
4          Don't know                No

[5 rows x 24 columns]
```

[4]:
```
gender = train_df['Gender'].str.lower()

gender = train_df['Gender'].unique()

male_str = ["male", "m", "male-ish", "maile", "mal", "male (cis)", "make",␣
↪"male ", "man","msle", "mail", "malr","cis man", "Cis Male", "cis male"]
trans_str = ["trans-female", "something kinda male?", "queer/she/they",␣
↪"non-binary","nah", "all", "enby", "fluid", "genderqueer", "androgyne",␣
↪"agender", "male leaning androgynous", "guy (-ish) ^_^", "trans woman",␣
↪"neuter", "female (trans)", "queer", "ostensibly male, unsure what that␣
↪really means"]
female_str = ["cis female", "f", "female", "woman",  "femake", "female␣
↪","cis-female/femme", "female (cis)", "femail"]

for (row, col) in train_df.iterrows():

    if str.lower(col.Gender) in male_str:
        train_df['Gender'].replace(to_replace=col.Gender, value='male',␣
↪inplace=True)

    if str.lower(col.Gender) in female_str:
        train_df['Gender'].replace(to_replace=col.Gender, value='female',␣
↪inplace=True)

    if str.lower(col.Gender) in trans_str:
        train_df['Gender'].replace(to_replace=col.Gender, value='trans',␣
↪inplace=True)

stk_list = ['A little about you', 'p']
train_df = train_df[~train_df['Gender'].isin(stk_list)]

print(train_df['Gender'].unique())
```

```
['female' 'male' 'trans']
```

[5]:
```
train_df['Age'].fillna(train_df['Age'].median(), inplace = True)

s = pd.Series(train_df['Age'])
s[s<18] = train_df['Age'].median()
```

```
train_df['Age'] = s
s = pd.Series(train_df['Age'])
s[s>120] = train_df['Age'].median()
train_df['Age'] = s

train_df['age_range'] = pd.cut(train_df['Age'], [0,20,30,65,100],␣
↪labels=["0-20", "21-30", "31-65", "66-100"], include_lowest=True)
```

[6]:
```
train_df['self_employed'] = train_df['self_employed'].replace([defaultString],␣
↪'No')
print(train_df['self_employed'].unique())
```

```
['No' 'Yes']
```

[7]:
```
train_df['work_interfere'] = train_df['work_interfere'].
↪replace([defaultString], 'Don\'t know' )
print(train_df['work_interfere'].unique())
```

```
['Often' 'Rarely' 'Never' 'Sometimes' "Don't know"]
```

Encoding Data

[8]:
```
labelDict = {}
for feature in train_df:
    le = preprocessing.LabelEncoder()
    le.fit(train_df[feature])
    le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    train_df[feature] = le.transform(train_df[feature])
    labelKey = 'label_' + feature
    labelValue = [*le_name_mapping]
    labelDict[labelKey] =labelValue

for key, value in labelDict.items():
    print(key, value)

train_df = train_df.drop(['Country'], axis= 1)
train_df.head()
```

```
label_Age [18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 53, 54, 55,
56, 57, 58, 60, 61, 62, 65, 72]
label_Gender ['female', 'male', 'trans']
label_Country ['Australia', 'Austria', 'Belgium', 'Bosnia and Herzegovina',
'Brazil', 'Bulgaria', 'Canada', 'China', 'Colombia', 'Costa Rica', 'Croatia',
'Czech Republic', 'Denmark', 'Finland', 'France', 'Georgia', 'Germany',
'Greece', 'Hungary', 'India', 'Ireland', 'Israel', 'Italy', 'Japan', 'Latvia',
'Mexico', 'Moldova', 'Netherlands', 'New Zealand', 'Nigeria', 'Norway',
'Philippines', 'Poland', 'Portugal', 'Romania', 'Russia', 'Singapore',
```

```
'Slovenia', 'South Africa', 'Spain', 'Sweden', 'Switzerland', 'Thailand',
'United Kingdom', 'United States', 'Uruguay', 'Zimbabwe']
label_self_employed ['No', 'Yes']
label_family_history ['No', 'Yes']
label_treatment ['No', 'Yes']
label_work_interfere ["Don't know", 'Never', 'Often', 'Rarely', 'Sometimes']
label_no_employees ['1-5', '100-500', '26-100', '500-1000', '6-25', 'More than
1000']
label_remote_work ['No', 'Yes']
label_tech_company ['No', 'Yes']
label_benefits ["Don't know", 'No', 'Yes']
label_care_options ['No', 'Not sure', 'Yes']
label_wellness_program ["Don't know", 'No', 'Yes']
label_seek_help ["Don't know", 'No', 'Yes']
label_anonymity ["Don't know", 'No', 'Yes']
label_leave ["Don't know", 'Somewhat difficult', 'Somewhat easy', 'Very
difficult', 'Very easy']
label_mental_health_consequence ['Maybe', 'No', 'Yes']
label_phys_health_consequence ['Maybe', 'No', 'Yes']
label_coworkers ['No', 'Some of them', 'Yes']
label_supervisor ['No', 'Some of them', 'Yes']
label_mental_health_interview ['Maybe', 'No', 'Yes']
label_phys_health_interview ['Maybe', 'No', 'Yes']
label_mental_vs_physical ["Don't know", 'No', 'Yes']
label_obs_consequence ['No', 'Yes']
label_age_range ['0-20', '21-30', '31-65', '66-100']
```

[8]:

	Age	Gender	self_employed	family_history	treatment	work_interfere	\
0	19	0	0	0	1	2	
1	26	1	0	0	0	3	
2	14	1	0	0	0	3	
3	13	1	0	1	1	2	
4	13	1	0	0	0	1	

	no_employees	remote_work	tech_company	benefits	…	leave	\
0	4	0	1	2	…	2	
1	5	0	0	0	…	0	
2	4	0	1	1	…	1	
3	2	0	1	1	…	1	
4	1	1	1	2	…	0	

	mental_health_consequence	phys_health_consequence	coworkers	supervisor	\
0	1	1	1	2	
1	0	1	0	0	
2	1	1	2	2	
3	2	2	1	0	
4	1	1	1	2	

```
   mental_health_interview  phys_health_interview  mental_vs_physical  \
0                        1                      0                   2
1                        1                      1                   0
2                        2                      2                   1
3                        0                      0                   1
4                        2                      2                   0

   obs_consequence  age_range
0                0          2
1                0          2
2                0          2
3                1          2
4                0          2

[5 rows x 24 columns]
```

[9]:

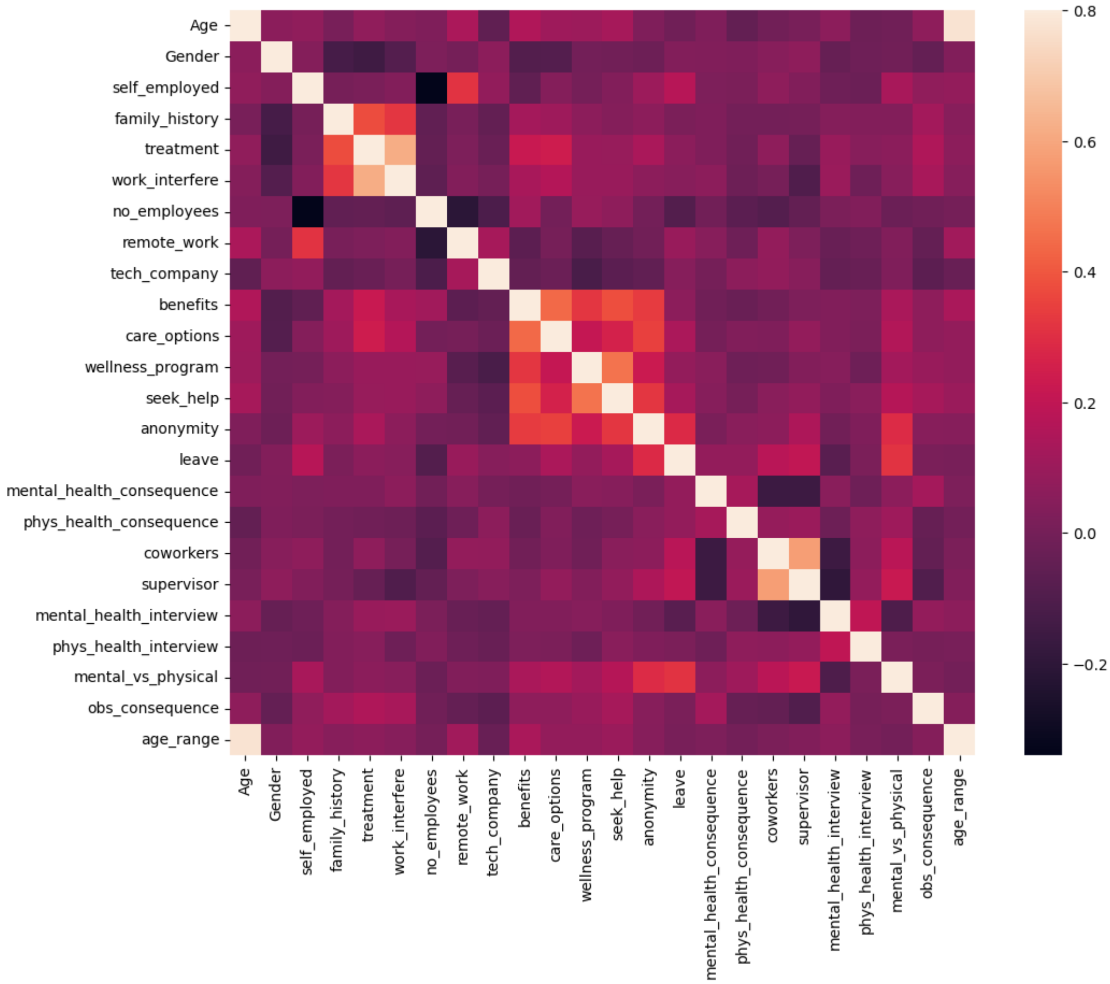
```
total = train_df.isnull().sum().sort_values(ascending=False)
percent = (train_df.isnull().sum()/train_df.isnull().count()).
 ↪sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
print(missing_data)
```

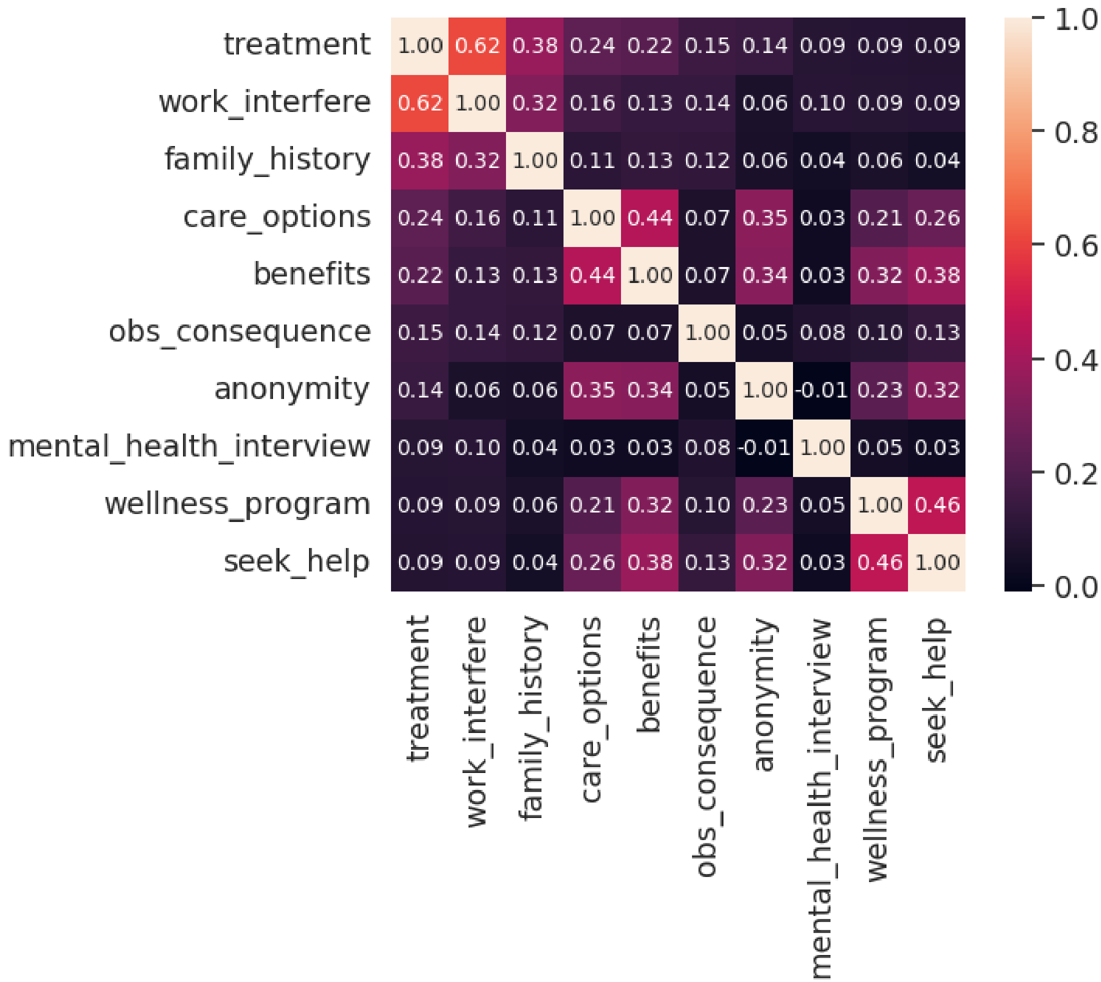
```
                           Total  Percent
Age                            0      0.0
Gender                         0      0.0
obs_consequence                0      0.0
mental_vs_physical             0      0.0
phys_health_interview          0      0.0
mental_health_interview        0      0.0
supervisor                     0      0.0
coworkers                      0      0.0
phys_health_consequence        0      0.0
mental_health_consequence      0      0.0
leave                          0      0.0
anonymity                      0      0.0
seek_help                      0      0.0
wellness_program               0      0.0
care_options                   0      0.0
benefits                       0      0.0
tech_company                   0      0.0
remote_work                    0      0.0
no_employees                   0      0.0
work_interfere                 0      0.0
treatment                      0      0.0
family_history                 0      0.0
```

```
self_employed                  0       0.0
age_range                      0       0.0
```

Covariance Matrix. Variability comparison between categories of variables

[10]:
```python
corrmat = train_df.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
plt.show()
k = 10
cols = corrmat.nlargest(k, 'treatment')['treatment'].index
cm = np.corrcoef(train_df[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yticklabels=cols.values, xticklabels=cols.values)
plt.show()
```





Scaling and Fitting

[11]:
```
scaler = MinMaxScaler()
train_df['Age'] = scaler.fit_transform(train_df[['Age']])
train_df.head()
```

[11]:
```
        Age  Gender  self_employed  family_history  treatment  work_interfere  \
0  0.431818       0              0               0          1               2
1  0.590909       1              0               0          0               3
2  0.318182       1              0               0          0               3
3  0.295455       1              0               1          1               2
4  0.295455       1              0               0          0               1

   no_employees  remote_work  tech_company  benefits  …  leave  \
0             4            0             1         2  …      2
1             5            0             0         0  …      0
2             4            0             1         1  …      1
```

```
3             2            0             1         1  …      1
4             1            1             1         2  …      0

   mental_health_consequence  phys_health_consequence  coworkers  supervisor  \
0                          1                        1          1           2
1                          0                        1          0           0
2                          1                        1          2           2
3                          2                        2          1           0
4                          1                        1          1           2

   mental_health_interview  phys_health_interview  mental_vs_physical  \
0                        1                      0                   2
1                        1                      1                   0
2                        2                      2                   1
3                        0                      0                   1
4                        2                      2                   0

   obs_consequence  age_range
0                0          2
1                0          2
2                0          2
3                1          2
4                0          2

[5 rows x 24 columns]
```

[12]:

```
feature_cols = ['Age', 'Gender', 'family_history', 'benefits', 'care_options',␣
 ↪'anonymity', 'leave', 'work_interfere']
X = train_df[feature_cols]
y = train_df.treatment
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
 ↪random_state=0)
methodDict = {}
rmseDict = ()
```

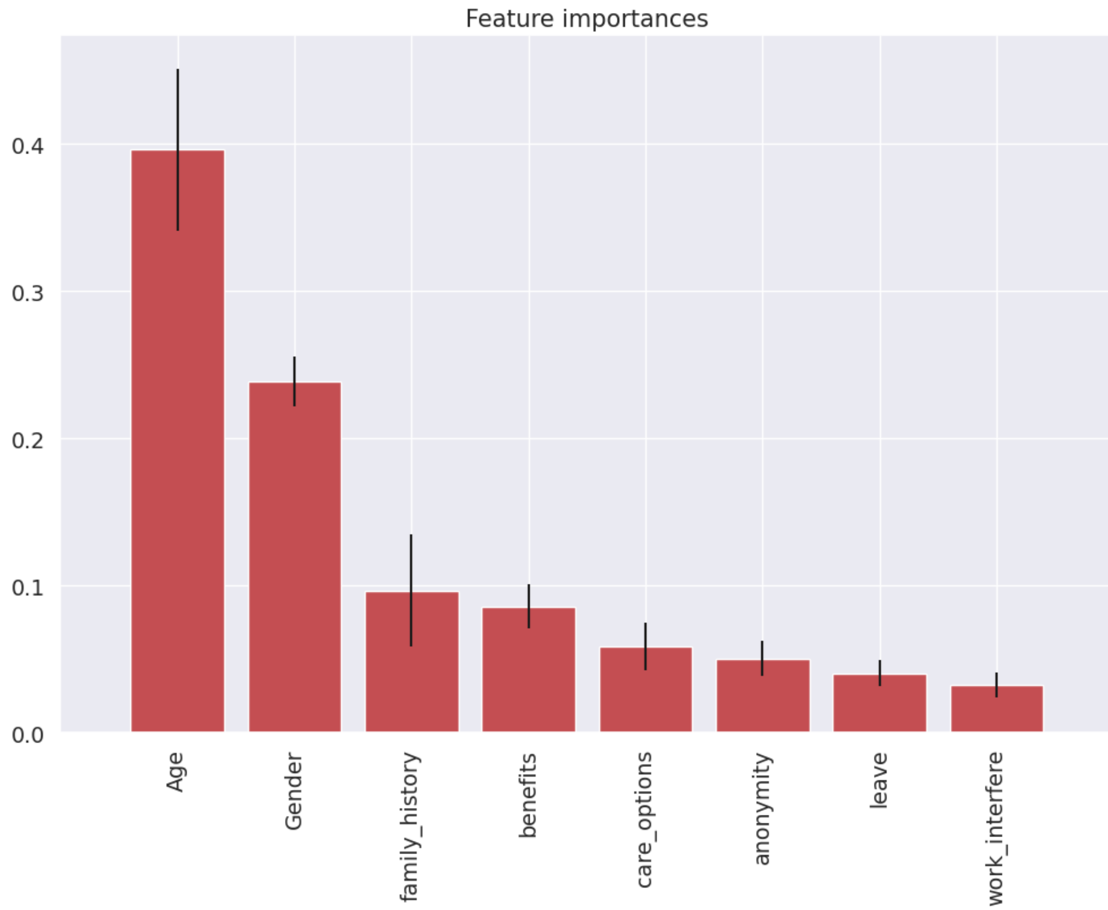
[13]:

```
forest = ExtraTreesClassifier(n_estimators=250,
                              random_state=0)

forest.fit(X, y)
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

labels = []
for f in range(X.shape[1]):
    labels.append(feature_cols[f])
```

```
plt.figure(figsize=(12,8))
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), labels, rotation='vertical')
plt.xlim([-1, X.shape[1]])
plt.show()
```



Tuning

```
[14]: def evalClassModel(model, y_test, y_pred_class, plot=False):
    print('Accuracy:', metrics.accuracy_score(y_test, y_pred_class))
    print('Null accuracy:\n', y_test.value_counts())
    print('Percentage of ones:', y_test.mean())
    print('Percentage of zeros:',1 - y_test.mean())
    print('True:', y_test.values[0:25])
    print('Pred:', y_pred_class[0:25])
```

```
    confusion = metrics.confusion_matrix(y_test, y_pred_class)
    TP = confusion[1, 1]
    TN = confusion[0, 0]
    FP = confusion[0, 1]
    FN = confusion[1, 0]

    sns.heatmap(confusion,annot=True,fmt="d")
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    accuracy = metrics.accuracy_score(y_test, y_pred_class)
    print('Classification Accuracy:', accuracy)
    print('Classification Error:', 1 - metrics.accuracy_score(y_test, y_pred_class))
    false_positive_rate = FP / float(TN + FP)
    print('False Positive Rate:', false_positive_rate)

    print('Precision:', metrics.precision_score(y_test, y_pred_class))
    print('AUC Score:', metrics.roc_auc_score(y_test, y_pred_class))
    print('Cross-validated AUC:', cross_val_score(model, X, y, cv=10, scoring='roc_auc').mean())
    print('First 10 predicted responses:\n', model.predict(X_test)[0:10])
    print('First 10 predicted probabilities of class members:\n', model.predict_proba(X_test)[0:10])

    model.predict_proba(X_test)[0:10, 1]
    y_pred_prob = model.predict_proba(X_test)[:, 1]

    if plot == True:
        plt.rcParams['font.size'] = 12
        plt.hist(y_pred_prob, bins=8)

        plt.xlim(0,1)
        plt.title('Histogram of predicted probabilities')
        plt.xlabel('Predicted probability of treatment')
        plt.ylabel('Frequency')

    y_pred_prob = y_pred_prob.reshape(-1,1)
    y_pred_class = binarize(y_pred_prob)[0]

    print('First 10 predicted probabilities:\n', y_pred_prob[0:10])

    roc_auc = metrics.roc_auc_score(y_test, y_pred_prob)

    fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob)
```

```
    if plot == True:
        plt.figure()

        plt.plot(fpr, tpr, color='darkorange', label='ROC curve (area = %0.2f)'␣
↪% roc_auc)
        plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.0])
        plt.rcParams['font.size'] = 12
        plt.title('ROC curve for treatment classifier')
        plt.xlabel('False Positive Rate (1 - Specificity)')
        plt.ylabel('True Positive Rate (Sensitivity)')
        plt.legend(loc="lower right")
        plt.show()

    def evaluate_threshold(threshold):

        print('Specificity for ' + str(threshold) + ' :', 1 - fpr[thresholds >␣
↪threshold][-1])

    predict_mine = np.where(y_pred_prob > 0.50, 1, 0)
    confusion = metrics.confusion_matrix(y_test, predict_mine)
    print(confusion)


    return accuracy
```

[15]:

```
def tuningCV(knn):

    k_range = list(range(1, 31))
    k_scores = []
    for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k)
        scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
        k_scores.append(scores.mean())
    print(k_scores)
    plt.plot(k_range, k_scores)
    plt.xlabel('Value of K for KNN')
    plt.ylabel('Cross-Validated Accuracy')
    plt.show()
```

[16]:

```
def tuningGridSerach(knn):
    k_range = list(range(1, 31))
    print(k_range)

    param_grid = dict(n_neighbors=k_range)
    print(param_grid)
```

```
    grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy')
    grid.fit(X, y)
    grid.grid_scores_

    print(grid.grid_scores_[0].parameters)
    print(grid.grid_scores_[0].cv_validation_scores)
    print(grid.grid_scores_[0].mean_validation_score)

    grid_mean_scores = [result.mean_validation_score for result in grid.
 ↪grid_scores_]
    print(grid_mean_scores)

    plt.plot(k_range, grid_mean_scores)
    plt.xlabel('Value of K for KNN')
    plt.ylabel('Cross-Validated Accuracy')
    plt.show()

    print('GridSearch best score', grid.best_score_)
    print('GridSearch best params', grid.best_params_)
    print('GridSearch best estimator', grid.best_estimator_)
```

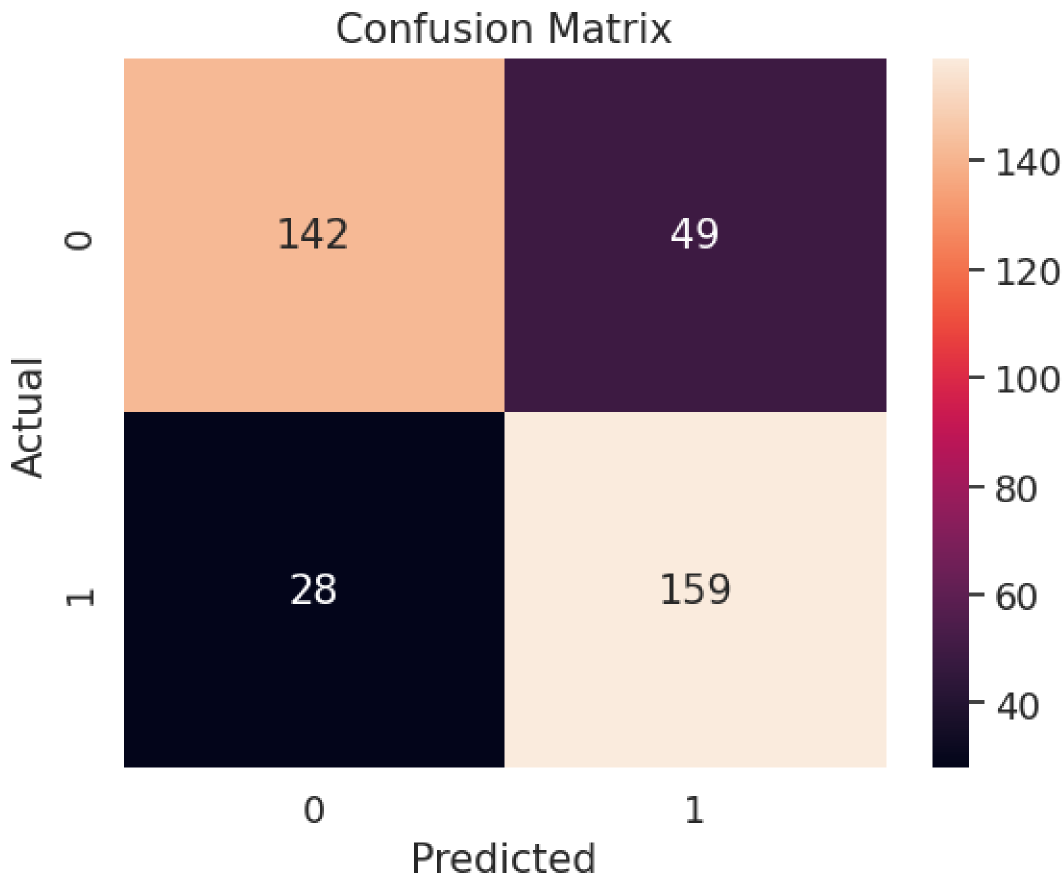
[17]:
```
def tuningRandomizedSearchCV(model, param_dist):
    rand = RandomizedSearchCV(model, param_dist, cv=10, scoring='accuracy',␣
 ↪n_iter=10, random_state=5)
    rand.fit(X, y)
    rand.cv_results_
    print('Rand. Best Score: ', rand.best_score_)
    print('Rand. Best Params: ', rand.best_params_)
    best_scores = []
    for _ in range(20):
        rand = RandomizedSearchCV(model, param_dist, cv=10, scoring='accuracy',␣
 ↪n_iter=10)
        rand.fit(X, y)
        best_scores.append(round(rand.best_score_, 3))
    print(best_scores)
```

Evaluating models

[18]:
```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred_class = logreg.predict(X_test)
accuracy_score = evalClassModel(logreg, y_test, y_pred_class, True)
methodDict['Log. Regres.'] = accuracy_score * 100
```

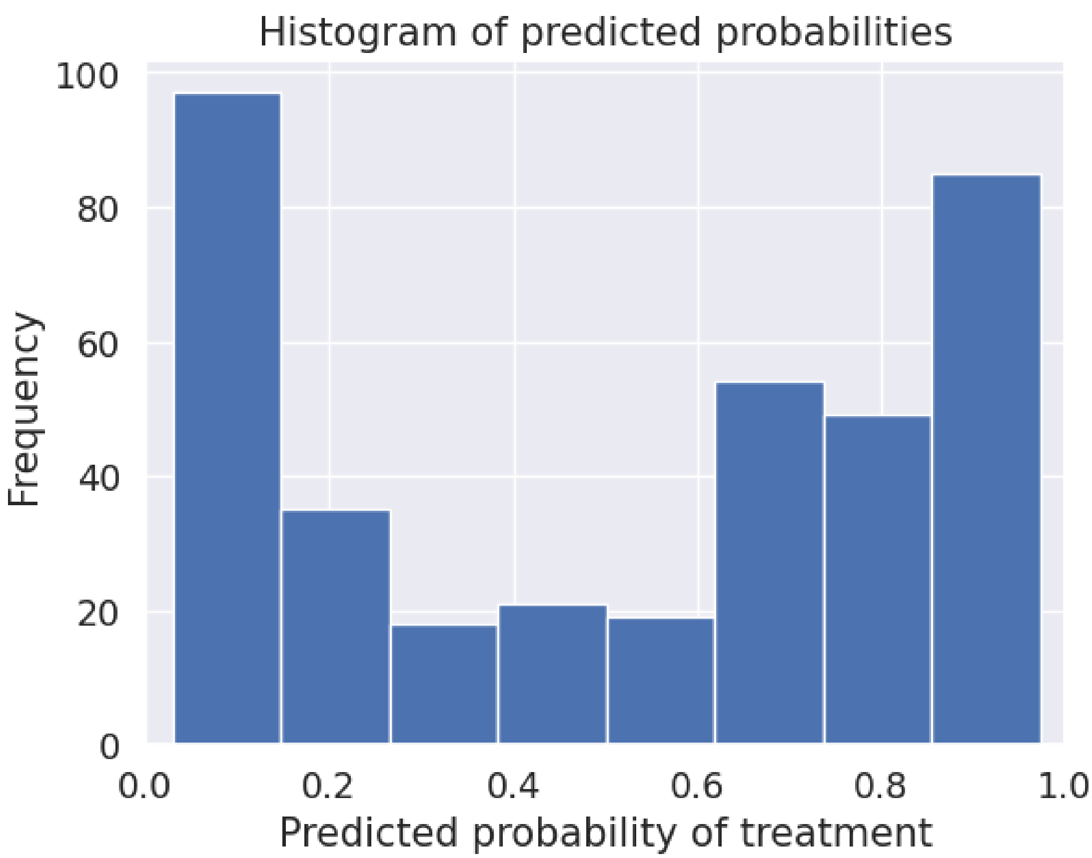
```
Accuracy: 0.7962962962962963
Null accuracy:
```

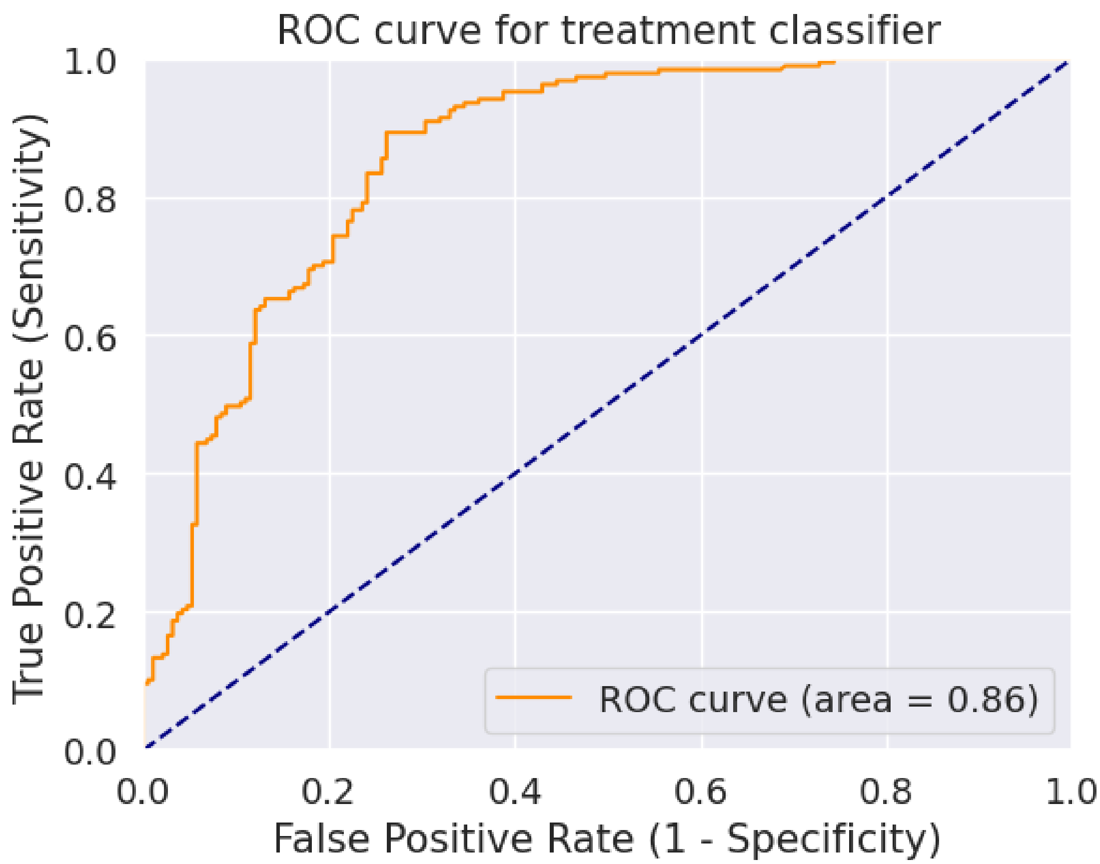
```
 treatment
0    191
1    187
Name: count, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 0]
```




```
Classification Accuracy: 0.7962962962962963
Classification Error: 0.20370370370370372
False Positive Rate: 0.25654450261780104
Precision: 0.7644230769230769
AUC Score: 0.7968614385306716
Cross-validated AUC: 0.8753623882722146
First 10 predicted responses:
 [1 0 0 0 1 1 0 1 0 1]
First 10 predicted probabilities of class members:
 [[0.09193053 0.90806947]
 [0.95991564 0.04008436]
```

```
 [0.96547467 0.03452533]
 [0.78757121 0.21242879]
 [0.38959922 0.61040078]
 [0.05264207 0.94735793]
 [0.75035574 0.24964426]
 [0.19065116 0.80934884]
 [0.61612081 0.38387919]
 [0.47699963 0.52300037]]
First 10 predicted probabilities:
 [[0.90806947]
 [0.04008436]
 [0.03452533]
 [0.21242879]
 [0.61040078]
 [0.94735793]
 [0.24964426]
 [0.80934884]
 [0.38387919]
 [0.52300037]]
```





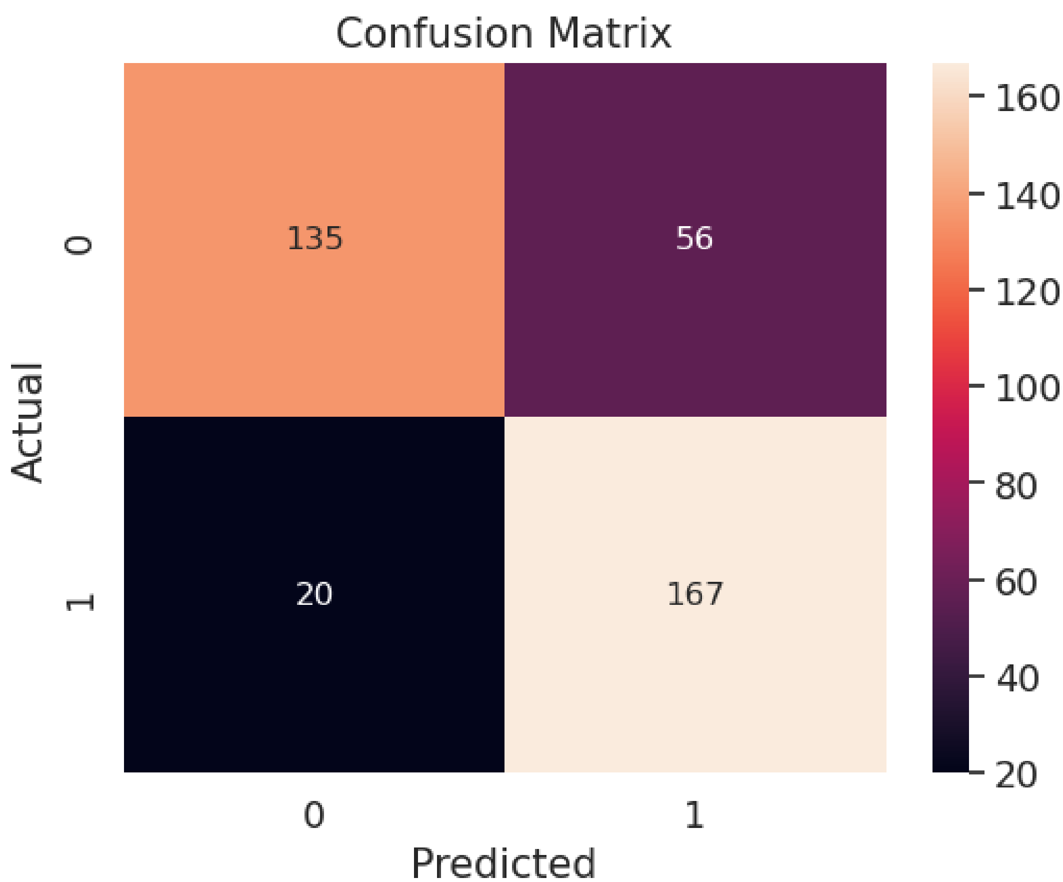

```
[[142  49]
 [ 28 159]]
```

[ ]:

[19]:
```
knn = KNeighborsClassifier(n_neighbors=5)
k_range = list(range(1, 31))
weight_options = ['uniform', 'distance']
param_dist = dict(n_neighbors=k_range, weights=weight_options)
tuningRandomizedSearchCV(knn, param_dist)
knn = KNeighborsClassifier(n_neighbors=27, weights='uniform')
knn.fit(X_train, y_train)
y_pred_class = knn.predict(X_test)
accuracy_score = evalClassModel(knn, y_test, y_pred_class, True)
methodDict['KNN'] = accuracy_score * 100
```

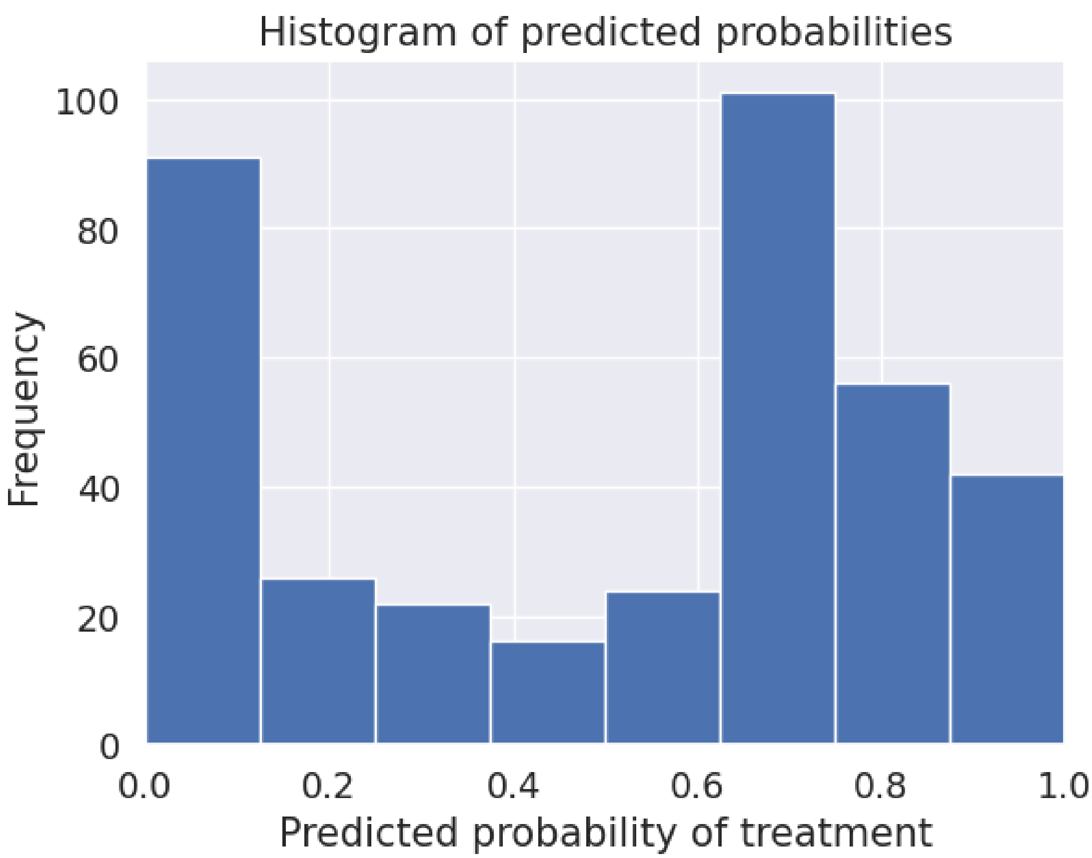
```
Rand. Best Score:  0.8201841269841269
Rand. Best Params:  {'weights': 'uniform', 'n_neighbors': 15}
[0.82, 0.822, 0.823, 0.823, 0.823, 0.816, 0.815, 0.819, 0.815, 0.822, 0.822,
0.815, 0.823, 0.823, 0.822, 0.815, 0.815, 0.815, 0.812, 0.815]
```

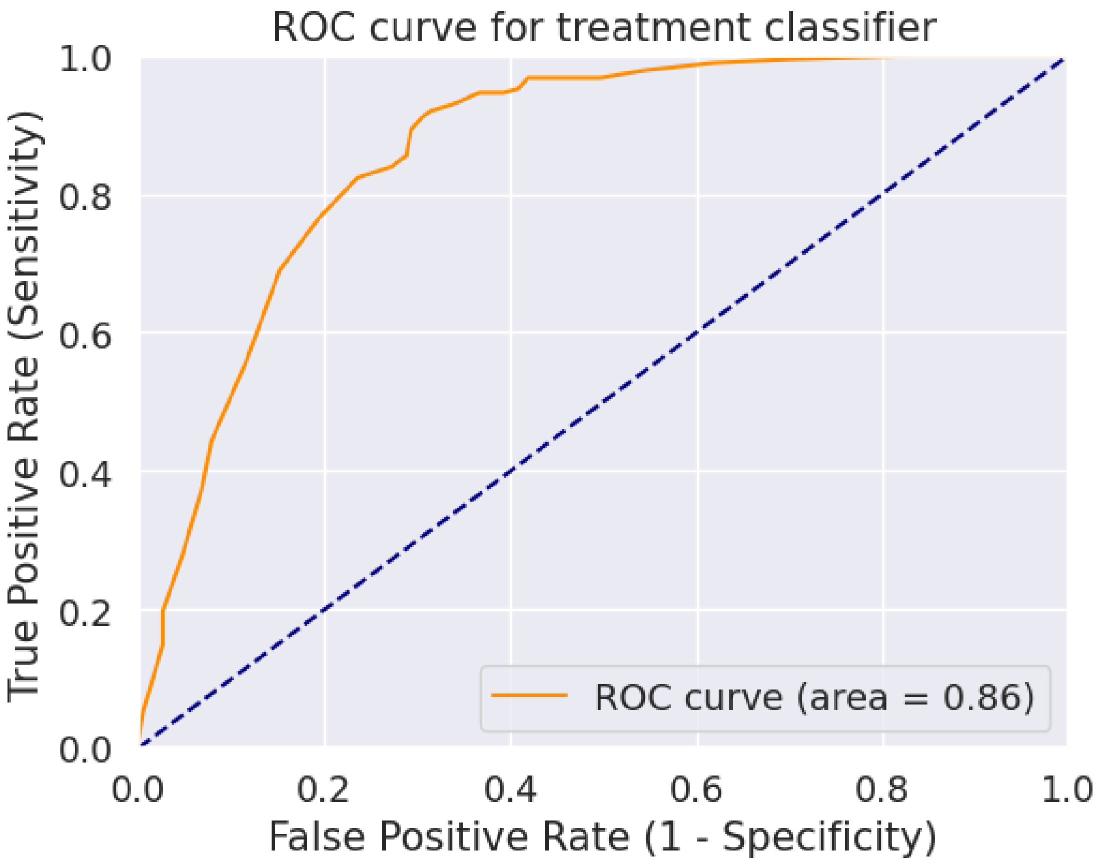
```
Accuracy: 0.798941798941799
Null accuracy:
 treatment
0    191
1    187
Name: count, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 0]
```




```
Classification Accuracy: 0.798941798941799
Classification Error: 0.20105820105820105
False Positive Rate: 0.2931937172774869
Precision: 0.7488789237668162
AUC Score: 0.7999272055323796
Cross-validated AUC: 0.8784682568890758
First 10 predicted responses:
 [1 0 0 0 1 1 0 1 1 1]
First 10 predicted probabilities of class members:
```

```
 [[0.33333333 0.66666667]
 [1.         0.        ]
 [1.         0.        ]
 [0.66666667 0.33333333]
 [0.37037037 0.62962963]
 [0.03703704 0.96296296]
 [0.59259259 0.40740741]
 [0.37037037 0.62962963]
 [0.33333333 0.66666667]
 [0.33333333 0.66666667]]
First 10 predicted probabilities:
 [[0.66666667]
 [0.        ]
 [0.        ]
 [0.33333333]
 [0.62962963]
 [0.96296296]
 [0.40740741]
 [0.62962963]
 [0.66666667]
 [0.66666667]]
```






```
[[135  56]
 [ 20 167]]
```

[ ]:

```

```

Predicting with Neural Network

[20]:

```
import tensorflow as tf
import argparse

batch_size = 100
train_steps = 1000

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
↪random_state=0)

def train_input_fn(features, labels, batch_size):
    dataset = tf.data.Dataset.from_tensor_slices((dict(features), labels))
```

```
    return dataset.shuffle(1000).repeat().batch(batch_size)

def eval_input_fn(features, labels, batch_size):
    features=dict(features)
    if labels is None:
        inputs = features
    else:
        inputs = (features, labels)

    dataset = tf.data.Dataset.from_tensor_slices(inputs)

    dataset = dataset.batch(batch_size)

    return dataset
```

[21]:
```
# Define Tensorflow feature columns
age = tf.feature_column.numeric_column("Age")
gender = tf.feature_column.numeric_column("Gender")
family_history = tf.feature_column.numeric_column("family_history")
benefits = tf.feature_column.numeric_column("benefits")
care_options = tf.feature_column.numeric_column("care_options")
anonymity = tf.feature_column.numeric_column("anonymity")
leave = tf.feature_column.numeric_column("leave")
work_interfere = tf.feature_column.numeric_column("work_interfere")
feature_columns = [age, gender, family_history, benefits, care_options,␣
  ↪anonymity, leave, work_interfere]
```

[22]:
```
# Build a DNN with 2 hidden layers and 10 nodes in each hidden layer.
model = tf.estimator.DNNClassifier(feature_columns=feature_columns,
                                    hidden_units=[10, 10],
                                    optimizer=tf.keras.optimizers.
  ↪Adam(learning_rate = 1e-2))
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: /tmp/tmp4q4j0mid
INFO:tensorflow:Using config: {'_model_dir': '/tmp/tmp4q4j0mid',
'_tf_random_seed': None, '_save_summary_steps': 100, '_save_checkpoints_steps':
None, '_save_checkpoints_secs': 600, '_session_config': allow_soft_placement:
true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_log_step_count_steps': 100, '_train_distribute': None, '_device_fn': None,
'_protocol': None, '_eval_distribute': None, '_experimental_distribute': None,
```

```
'_experimental_max_worker_delay_secs': None, '_session_creation_timeout_secs':
7200, '_checkpoint_save_graph_def': True, '_service': None, '_cluster_spec':
ClusterSpec({}), '_task_type': 'worker', '_task_id': 0, '_global_id_in_cluster':
0, '_master': '', '_evaluation_master': '', '_is_chief': True,
'_num_ps_replicas': 0, '_num_worker_replicas': 1}
```

[23]:
```python
model.train(input_fn=lambda:train_input_fn(X_train, y_train, batch_size),␣
 ↪steps=train_steps)
```

```
WARNING:tensorflow:From /opt/software/lib/python3.10/site-
packages/tensorflow/python/training/training_util.py:396:
Variable.initialized_value (from tensorflow.python.ops.variables) is deprecated
and will be removed in a future version.
Instructions for updating:
Use Variable.read_value. Variables in 2.X are initialized automatically both in
eager and graph (inside tf.defun) contexts.
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
```

```
2023-12-20 23:13:50.147118: I tensorflow/core/platform/cpu_feature_guard.cc:151]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

```
INFO:tensorflow:Calling checkpoint listeners before saving checkpoint 0…
INFO:tensorflow:Saving checkpoints for 0 into /tmp/tmp4q4j0mid/model.ckpt.
INFO:tensorflow:Calling checkpoint listeners after saving checkpoint 0…
INFO:tensorflow:loss = 0.73677033, step = 0
INFO:tensorflow:global_step/sec: 438.193
INFO:tensorflow:loss = 0.39001012, step = 100 (0.230 sec)
INFO:tensorflow:global_step/sec: 622.839
INFO:tensorflow:loss = 0.470975, step = 200 (0.160 sec)
INFO:tensorflow:global_step/sec: 627.507
INFO:tensorflow:loss = 0.3817996, step = 300 (0.159 sec)
INFO:tensorflow:global_step/sec: 593.106
INFO:tensorflow:loss = 0.3984199, step = 400 (0.168 sec)
INFO:tensorflow:global_step/sec: 585.428
INFO:tensorflow:loss = 0.4657975, step = 500 (0.171 sec)
INFO:tensorflow:global_step/sec: 605.726
INFO:tensorflow:loss = 0.33196712, step = 600 (0.165 sec)
INFO:tensorflow:global_step/sec: 600.106
INFO:tensorflow:loss = 0.30280408, step = 700 (0.166 sec)
INFO:tensorflow:global_step/sec: 589.153
```

```
INFO:tensorflow:loss = 0.20976566, step = 800 (0.170 sec)
INFO:tensorflow:global_step/sec: 598.269
INFO:tensorflow:loss = 0.30433273, step = 900 (0.167 sec)
INFO:tensorflow:Calling checkpoint listeners before saving checkpoint 1000…
INFO:tensorflow:Saving checkpoints for 1000 into /tmp/tmp4q4j0mid/model.ckpt.
INFO:tensorflow:Calling checkpoint listeners after saving checkpoint 1000…
INFO:tensorflow:Loss for final step: 0.31433606.
```

[23]:
```
<tensorflow_estimator.python.estimator.canned.dnn.DNNClassifierV2 at
0x7f2a2afdbc70>
```

[24]:
```
eval_result = model.evaluate(
    input_fn=lambda:eval_input_fn(X_test, y_test, batch_size))
print('\nTest set accuracy: {accuracy:0.2f}\n'.format(**eval_result))
accuracy = eval_result['accuracy'] * 100
methodDict['NN DNNClasif.'] = accuracy
```

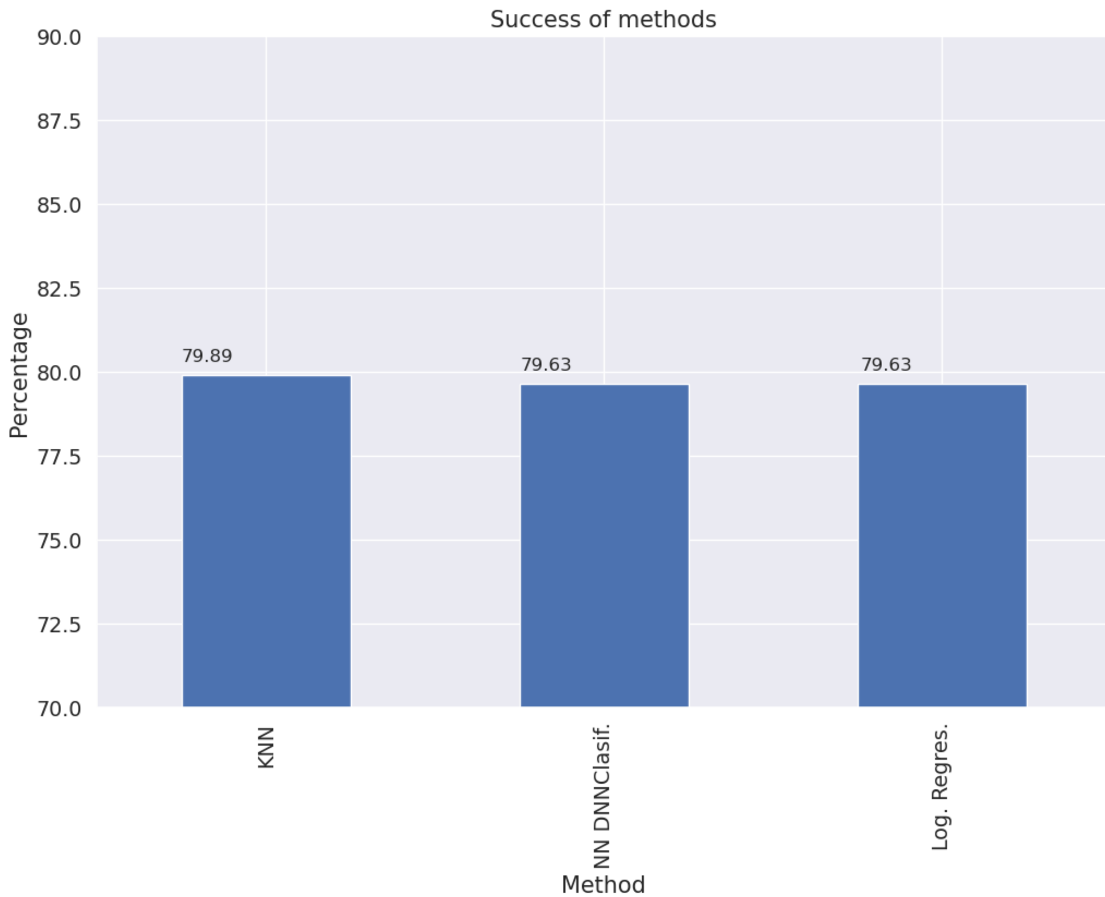
```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2023-12-20T23:13:52
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from /tmp/tmp4q4j0mid/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Inference Time : 0.72190s
INFO:tensorflow:Finished evaluation at 2023-12-20-23:13:53
INFO:tensorflow:Saving dict for global step 1000: accuracy = 0.7962963,
accuracy_baseline = 0.505291, auc = 0.8791052, auc_precision_recall =
0.85526246, average_loss = 0.4675138, global_step = 1000, label/mean =
0.49470899, loss = 0.46714175, precision = 0.7570093, prediction/mean =
0.5003519, recall = 0.8663102
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 1000:
/tmp/tmp4q4j0mid/model.ckpt-1000

Test set accuracy: 0.80
```

Success method plot

[25]:
```
s = pd.Series(methodDict)
s = s.sort_values(ascending=False)
plt.figure(figsize=(12,8))
ax = s.plot(kind='bar')
for p in ax.patches:
    ax.annotate(str(round(p.get_height(),2)), (p.get_x() * 1.005, p.
 ↪get_height() * 1.005))
plt.ylim([70.0, 90.0])
plt.xlabel('Method')
```

```
plt.ylabel('Percentage')
plt.title('Success of methods')
plt.show()
```



[ ]:

[ ]:

[ ]:

[ ]: