Classifying chronic kidney disease (CKD) using a machine learning algorithm involves several steps, including data preprocessing, feature selection, model selection, training, and evaluation. Below is a general outline of the process:

1. **Data Collection and Understanding:**
   - Acquire a dataset that includes relevant features related to chronic kidney disease. Common features include age, blood pressure, serum creatinine, and other clinical measurements.
   - Understand the characteristics and distribution of the dataset.
2. **Data Preprocessing:**
   - Handle missing values: Impute or remove missing data.
   - Encode categorical variables: Convert categorical variables into numerical representations (e.g., one-hot encoding).
   - Normalize/Standardize numerical features: Bring all features to a similar scale.
3. **Feature Selection:**
   - Identify and select relevant features. Feature selection can improve model performance and reduce overfitting.
   - Techniques include correlation analysis, recursive feature elimination, or feature importance from tree-based models.
4. **Data Splitting:**
   - Split the dataset into training and testing sets to evaluate the model's performance.
5. **Model Selection:**
   - Choose a suitable machine learning algorithm for classification. Common algorithms for classification tasks include Decision Trees, Random Forest, Support Vector Machines, Logistic Regression, or Gradient Boosting.
6. **Model Training:**

- Train the selected model on the training dataset using the chosen features.

7. **Model Evaluation:**
   - Evaluate the model's performance on the testing set, using metrics such as accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC).
   - Adjust hyperparameters to improve performance if needed.
8. **Fine-tuning and Optimization:**
   - Perform hyperparameter tuning to optimize the model's performance.
   - Consider using techniques like cross-validation for more robust evaluation.
9. **Model Interpretation (Optional):**
   - If interpretability is crucial, use techniques such as SHAP values or feature importance to understand the model's decision-making process.
10. **Deployment:**
    - Once satisfied with the model's performance, deploy it for making predictions on new data.

Predicting Chronic Kidney Disease based on health records

Given 24 health related attributes taken in 2-month period of 400 patients, using the information of the 158 patients with complete records to predict the outcome (i.e. whether one has chronic kidney disease) of the remaining 242 patients (with missing values in their records).

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import roc_curve, auc, confusion_matrix, classification_report,accuracy_score
```

```
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

def auc_scorer(clf, X, y, model): # Helper function to plot th
e ROC curve
    if model=='RF':
        fpr, tpr, _ = roc_curve(y, clf.predict_proba(X)[:,1])
    elif model=='SVM':
        fpr, tpr, _ = roc_curve(y, clf.decision_function(X))
    roc_auc = auc(fpr, tpr)

     plt.figure()    # Plot the ROC curve
    plt.plot(fpr, tpr, label='ROC curve from '+model+' model (
area = %0.3f)' % roc_auc)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend(loc="lower right")
    plt.show()

    return fpr,tpr,roc_auc

# from subprocess import check_output
# print(check_output(["ls", "../input"]).decode("utf8"))
```

Load files

```
df = pd.read_csv('../input/kidney_disease.csv')
```

Cleaning and preprocessing of data for training a classifier

```
# Map text to 1/0 and do some cleaning
df[['htn','dm','cad','pe','ane']] = df[['htn','dm','cad','pe',
'ane']].replace(to_replace={'yes':1,'no':0})
df[['rbc','pc']] = df[['rbc','pc']].replace(to_replace={'abnor
mal':1,'normal':0})
df[['pcc','ba']] = df[['pcc','ba']].replace(to_replace={'prese
nt':1,'notpresent':0})
df[['appet']] = df[['appet']].replace(to_replace={'good':1,'po
or':0,'no':np.nan})
df['classification'] = df['classification'].replace(to_replace
={'ckd':1.0,'ckd\t':1.0,'notckd':0.0,'no':0.0})
```

```
df.rename(columns={'classification':'class'},inplace=True)
# Further cleaning
df['pe'] = df['pe'].replace(to_replace='good',value=0) # Not h
aving pedal edema is good
df['appet'] = df['appet'].replace(to_replace='no',value=0)
df['cad'] = df['cad'].replace(to_replace='\tno',value=0)
df['dm'] = df['dm'].replace(to_replace={'\tno':0,'\tyes':1,' y
es':1, '':np.nan})
df.drop('id',axis=1,inplace=True)
df.head()
```

Check the portion of rows with NaN

- Now the data is cleaned with improper values labelled NaN. Let's see how many NaNs are there.
- Drop all the rows with NaN values, and build a model out of this dataset (i.e. df2)
  - `df2 = df.dropna(axis=0)`
  - `df2['class'].value_counts()`

  `Out[6]:`
  - `0.0    115`
  - `1.0     43`
  - `Name: class, dtype:`

Examine correlations between different features

```
corr_df = df2.corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr_df, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_df, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5
})
plt.title('Correlations between different predictors')
plt.show()
```

Split the set for training models further into a (sub-)training set and testing set

```
X_train, X_test, y_train, y_test = train_test_split(df2.iloc[:,:-1], df2['class'],  test_size = 0.33, random,stratify= df2['class']
```

Choosing parameters with GridSearchCV with 10-fold cross validations.

```
tuned_parameters = [{'n_estimators':[7,8,9,10,11,12,13,14,15,16],'max_depth':[2,3,4,5,6,None],
                     'class_weight':[None,{0: 0.33,1:0.67},'balanced'],'random_state':[42]}]
clf = GridSearchCV(RandomForestClassifier(), tuned_parameters, cv=10,scoring='f1')
clf.fit(X_train, y_train)

print("Detailed classification report:")
y_true, lr_pred = y_test, clf.predict(X_test)
print(classification_report(y_true, lr_pred))

confusion = confusion_matrix(y_test, lr_pred)
print('Confusion Matrix:')
print(confusion)

# Determine the false positive and true positive rates
fpr,tpr,roc_auc = auc_scorer(clf, X_test, y_test, 'RF')

print('Best parameters:')
print(clf.best_params_)
clf_best = clf.best_estimator_
```

Examine feature importance

```
plt.figure(figsize=(12,3))
features = X_test.columns.values.tolist()
importance = clf_best.feature_importances_.tolist()
feature_series = pd.Series(data=importance,index=features)
feature_series.plot.bar()
plt.title('Feature Importance')
```

CONCLUSION:

Random forest

An ML-based disease classification system using the UCI CKD dataset was employed in [10]. Out of all implemented algorithms, random forest achieved the highest accuracy value.