supp_mat

April 15, 2020

1 Supplemental Materials

[56]:
```
import pkg_resources
if pkg_resources.get_distribution("scikit-learn").version == '0.21.3':
    !pip install --upgrade scikit-learn scipy pandas seaborn
```

```
Requirement already up-to-date: scikit-learn in /opt/conda/lib/python3.7/site-
packages (0.22.2.post1)
Requirement already up-to-date: scipy in /opt/conda/lib/python3.7/site-packages
(1.4.1)
Requirement already up-to-date: pandas in /opt/conda/lib/python3.7/site-packages
(1.0.3)
Requirement already up-to-date: seaborn in /opt/conda/lib/python3.7/site-
packages (0.10.0)
Requirement already satisfied, skipping upgrade: joblib>=0.11 in
/opt/conda/lib/python3.7/site-packages (from scikit-learn) (0.14.0)
Requirement already satisfied, skipping upgrade: numpy>=1.11.0 in
/opt/conda/lib/python3.7/site-packages (from scikit-learn) (1.17.2)
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in
/opt/conda/lib/python3.7/site-packages (from pandas) (2019.3)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.6.1 in
/opt/conda/lib/python3.7/site-packages (from pandas) (2.8.0)
Requirement already satisfied, skipping upgrade: matplotlib>=2.1.2 in
/opt/conda/lib/python3.7/site-packages (from seaborn) (3.1.1)
Requirement already satisfied, skipping upgrade: six>=1.5 in
/opt/conda/lib/python3.7/site-packages (from python-dateutil>=2.6.1->pandas)
(1.12.0)
Requirement already satisfied, skipping upgrade:
pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /opt/conda/lib/python3.7/site-
packages (from matplotlib>=2.1.2->seaborn) (2.4.2)
Requirement already satisfied, skipping upgrade: cycler>=0.10 in
/opt/conda/lib/python3.7/site-packages (from matplotlib>=2.1.2->seaborn)
(0.10.0)
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.7/site-packages (from matplotlib>=2.1.2->seaborn) (1.1.0)
Requirement already satisfied, skipping upgrade: setuptools in
/opt/conda/lib/python3.7/site-packages (from
```

```
kiwisolver>=1.0.1->matplotlib>=2.1.2->seaborn) (41.0.1)
```

[2]:
```
import warnings
warnings.filterwarnings("ignore")
```

[3]:
```
# Display plots inline
%matplotlib inline

# Data libraries
import pandas as pd
import numpy as np

# Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting defaults
plt.rcParams['figure.figsize'] = (8,5)
plt.rcParams['figure.dpi'] = 80

# ipython interactive widgets
from ipywidgets import interact

# sklearn modules
import sklearn
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LogisticRegression
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix
import sklearn.tree
from sklearn.model_selection import GridSearchCV, KFold
import sklearn.ensemble
import sklearn.neighbors
import sklearn.preprocessing
from sklearn.pipeline import make_pipeline
from scipy import stats
```

[4]:
```
wine_train = pd.read_csv("wine_qual_train.csv")
wine_test  = pd.read_csv("wine_qual_test.csv")
```

[8]:
```
def hide_current_axis(*args, **kwds):
    plt.gca().set_visible(False)

g = sns.pairplot(wine_train, hue='wine')
g.map_upper(hide_current_axis)
```

[8]: <seaborn.axisgrid.PairGrid at 0x7f6a65765cf8>

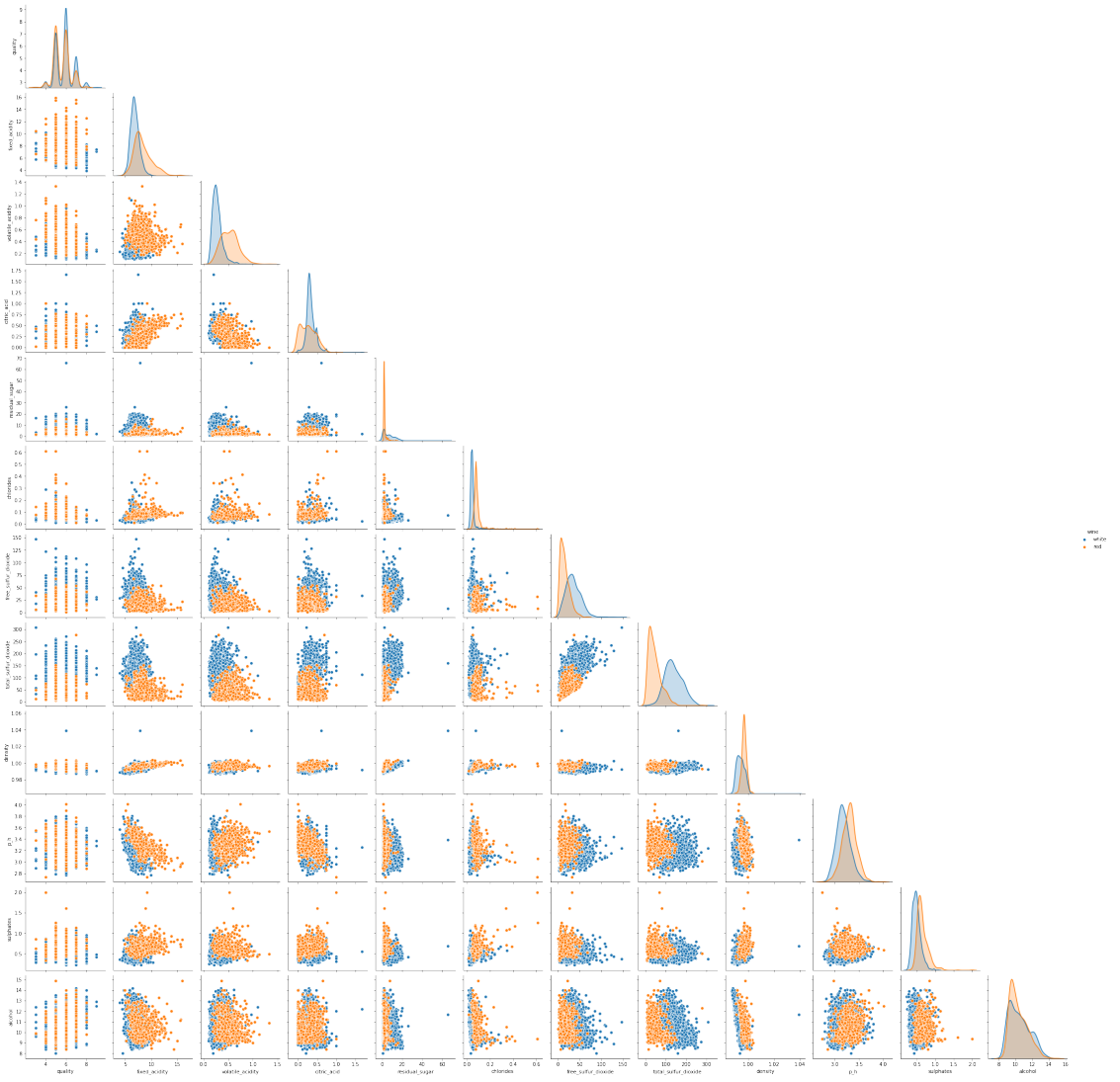


Figure 1: Pairplot of data

[5]:
```
wine_train['wine_white'] = wine_train['wine'].isin(['white']).astype(int)
wine_test['wine_white'] = wine_test['wine'].isin(['white']).astype(int)
```

[6]:
```
wine_train = wine_train.drop(['wine'], axis=1)
wine_test = wine_test.drop(['wine'], axis=1)
```

[21]:
```
corr = abs(wine_train.corr())
mask = np.triu(np.ones_like(corr, dtype=np.bool))
f, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(corr, mask=mask, annot=True)
```

```
b, t = plt.ylim()
b += 0.5
t -= 0.5
plt.ylim(b, t)
plt.show()
```

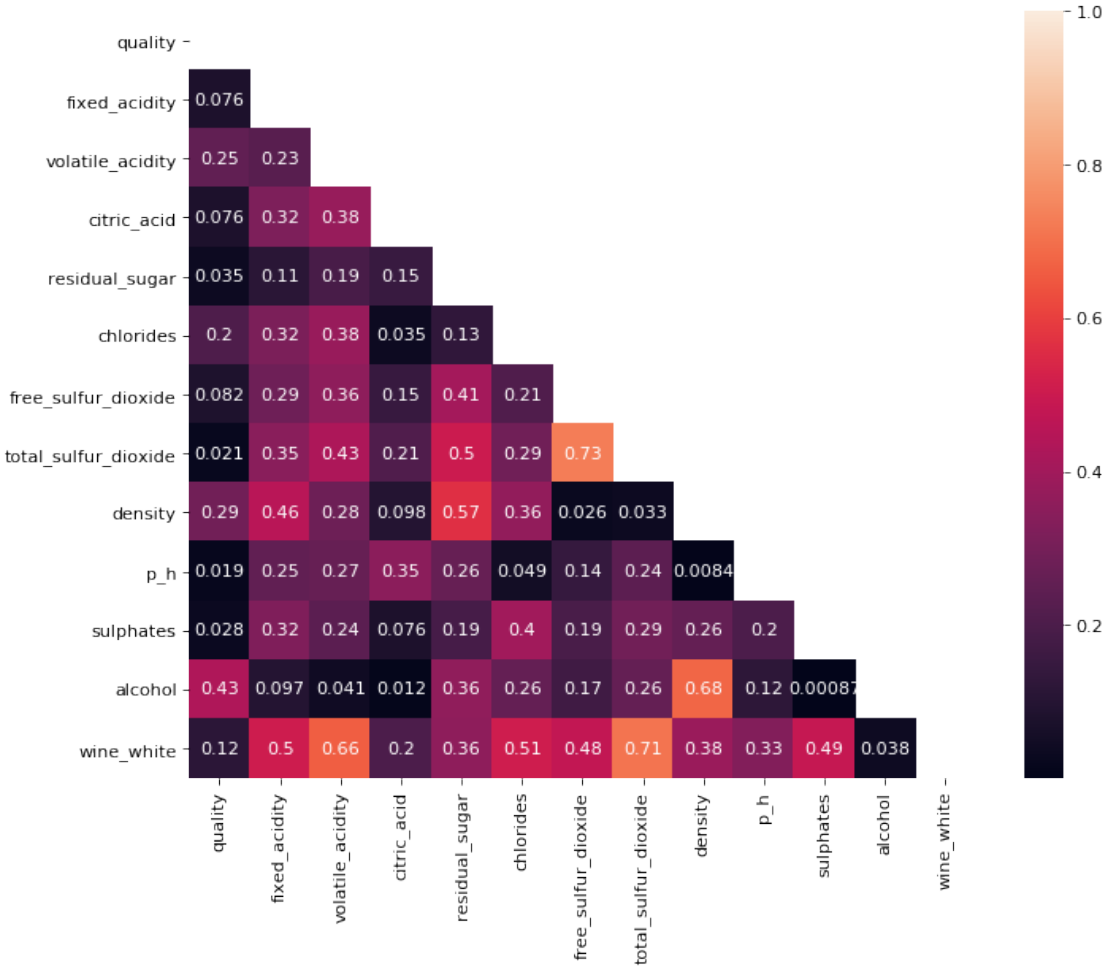
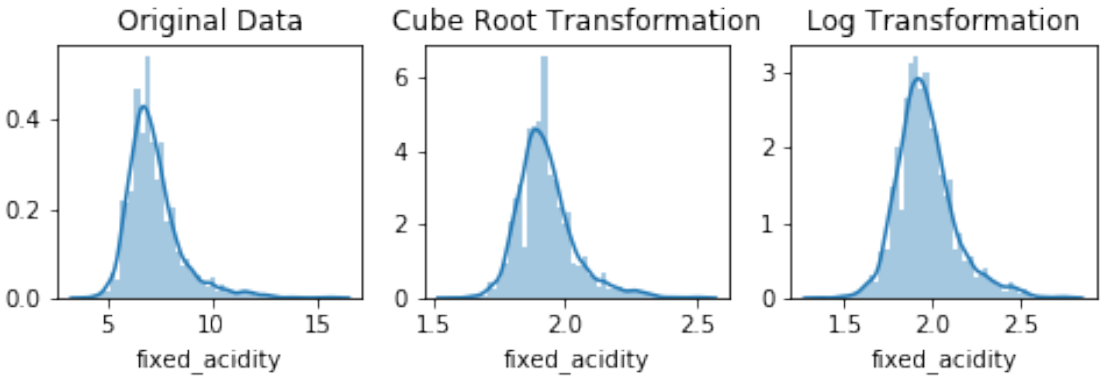
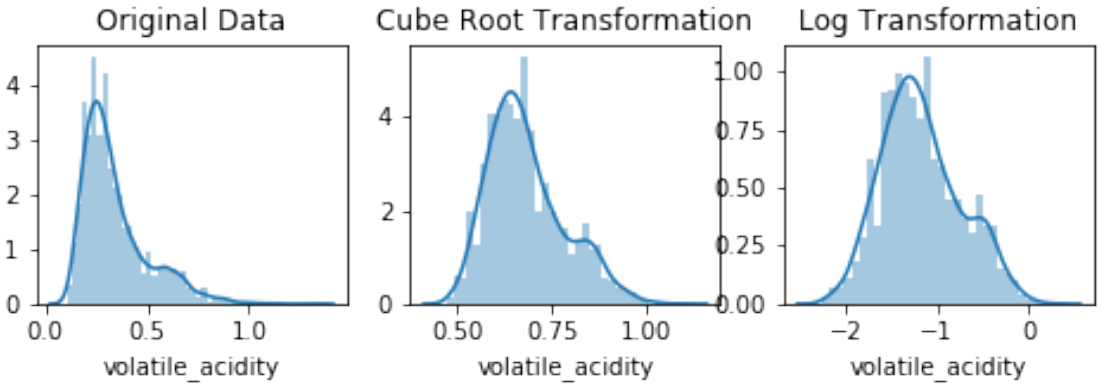


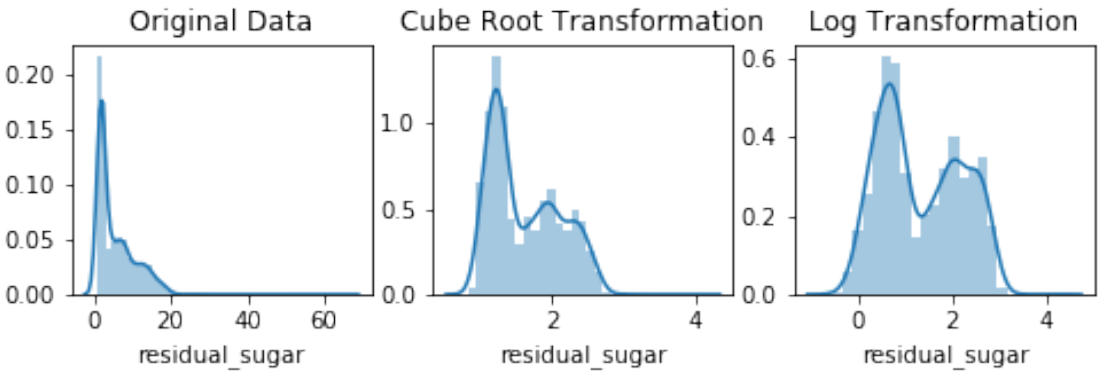
Figure 2: Correlation Plot Before Data Transformations

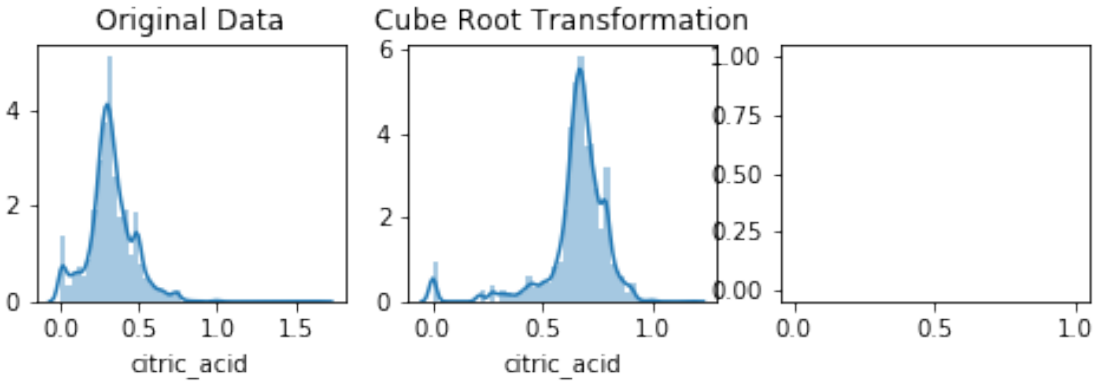
[7]:
```
for i in ['fixed_acidity', 'volatile_acidity', 'residual_sugar', 'citric_acid',␣
↪'chlorides', 'free_sulfur_dioxide', 'density', 'sulphates']:
    if i != 'quality' and i != 'wine_red' and i != 'wine_white':
        try:
            fig = plt.figure(figsize=(8,2))
            plt.subplot(131)
            sns.distplot(wine_train[i])
```

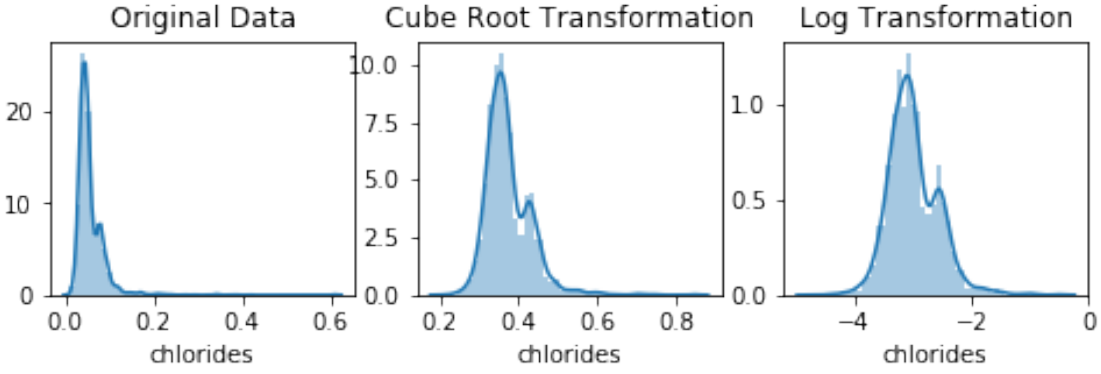
```
            plt.title('Original Data')
            plt.subplot(132)
            sns.distplot((wine_train[i])**(1/3))
            plt.title('Cube Root Transformation')
            plt.subplot(133)
            sns.distplot(np.log(wine_train[i]))
            plt.title('Log Transformation')
        except:
            print(' ')
    plt.show()
```

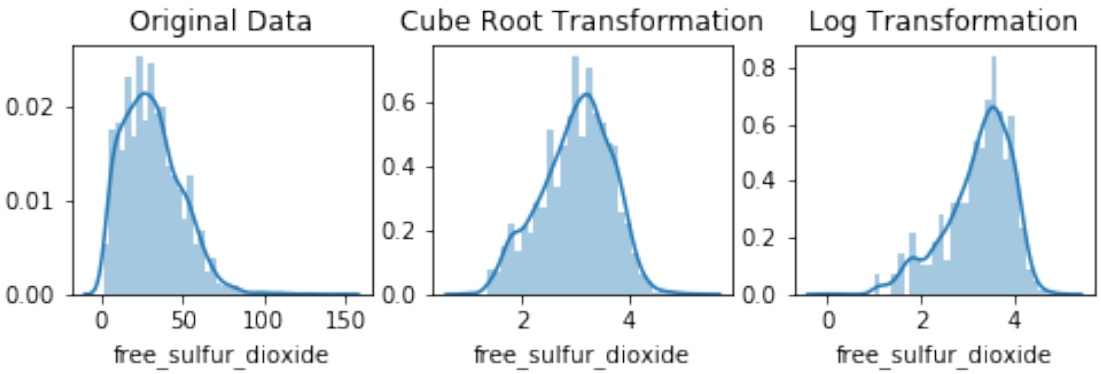


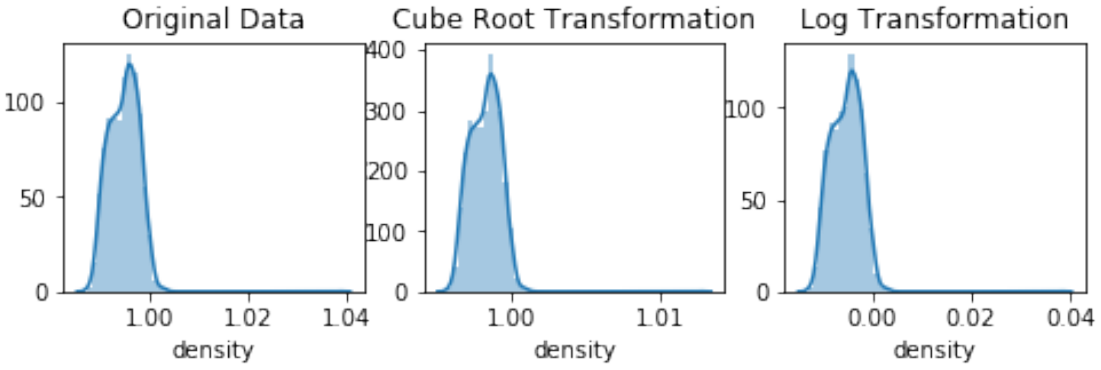


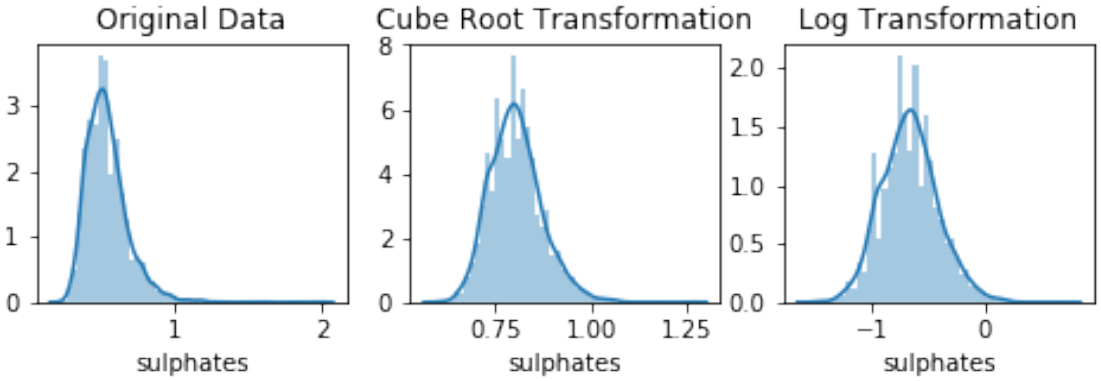

Original Data
Cube Root Transformation
Log Transformation
residual_sugar
residual_sugar
residual_sugar


Original Data
Cube Root Transformation
citric_acid
citric_acid


Original Data
Cube Root Transformation
Log Transformation
chlorides
chlorides
chlorides


Original Data
Cube Root Transformation
Log Transformation
0.02
0.01
0.00
0.6
0.4
0.2
0.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
2
4
0
2
4
free_sulfur_dioxide
free_sulfur_dioxide
free_sulfur_dioxide


Original Data
Cube Root Transformation
Log Transformation
100
50
0
400
300
200
100
0
100
50
0
1.00
1.02
1.04
1.00
1.01
0.00
0.02
0.04
density
density
density

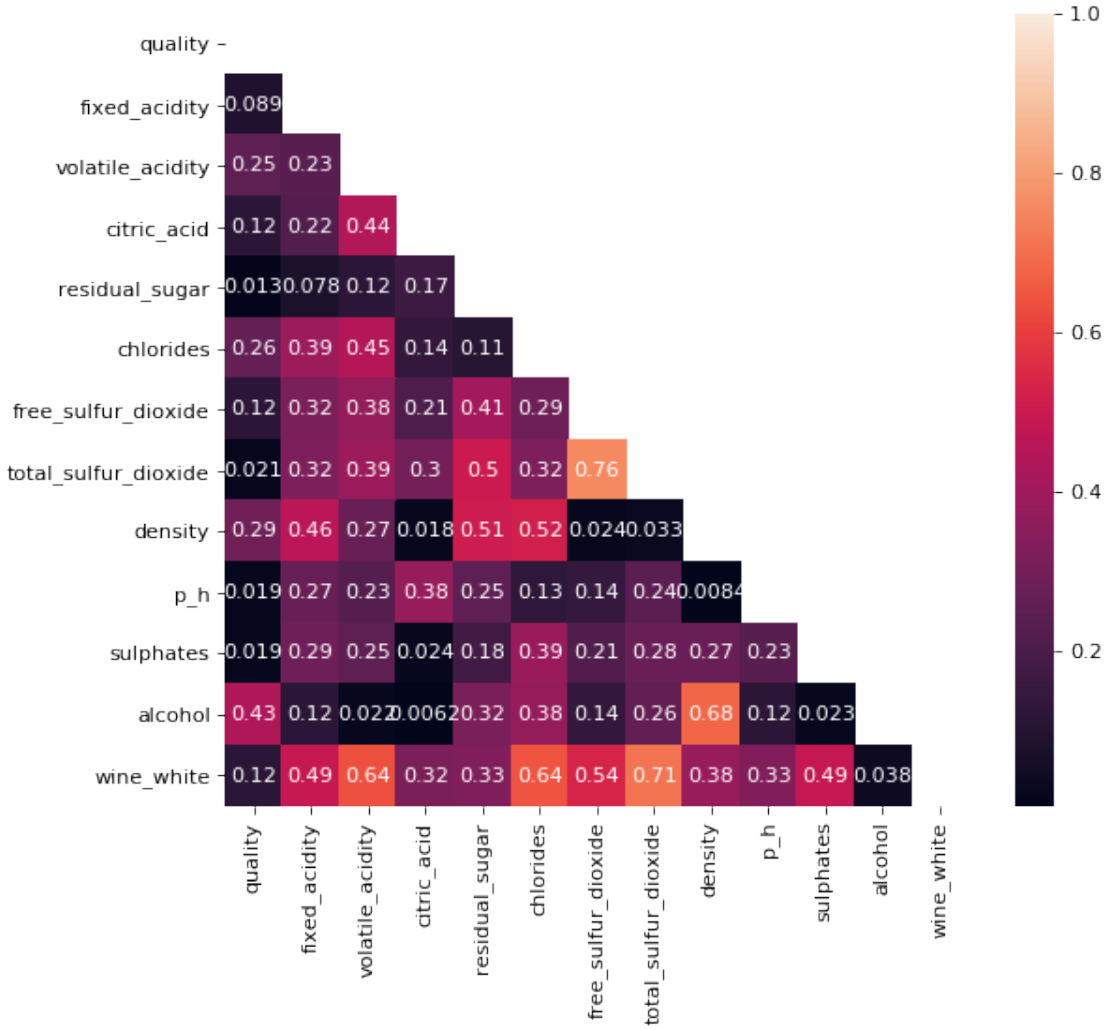

Original Data
Cube Root Transformation
Log Transformation
3
2
1
0
8
6
4
2
0
2.0
1.5
1.0
0.5
0.0
1
2
0.75
1.00
1.25
−1
0
sulphates
sulphates
sulphates

1.0.1 Figure 3: Distribution Plots Testing Different Data Transformations

[8]:
```
def df_transform(df):
    for col in ['fixed_acidity', 'volatile_acidity', 'residual_sugar',␣
 ↪'chlorides', 'sulphates']:
        df[col] = np.log(df[col])
    for col2 in ['free_sulfur_dioxide', 'citric_acid']:
        df[col2] = (df[col2])**(1/3)
```

[9]:
```
df_transform(wine_train)
df_transform(wine_test)
```

[25]:
```
corr = abs(wine_train.corr())
mask = np.triu(np.ones_like(corr, dtype=np.bool))
sns.heatmap(corr, mask=mask, annot=True)

b, t = plt.ylim()
b += 0.5
t -= 0.5
plt.ylim(b, t)
plt.show()
```



1.0.2 Figure 4: Correlation Plot after Data Transformations

[10]:
```
def quality_to_cat(df, cat):
    for i in range(len(df)):
        if df[cat][i] >= 7:
            df[cat][i] = 'Excellent'
        elif df[cat][i] == 6:
            df[cat][i] = 'Good'
        elif df[cat][i] == 5:
            df[cat][i] = 'Average'
        elif df[cat][i] <= 4:
            df[cat][i] = 'Poor'

quality_to_cat(wine_train, 'quality')
```

```
quality_to_cat(wine_test, 'quality')
```

[11]:
```
# Training data
Xt = wine_train.drop(['quality'], axis=1)
yt = wine_train.quality

print('Xt:', Xt.shape)
print('yt:', yt.shape)

# Validation data
Xv = wine_test.drop(['quality'], axis=1)
yv = wine_test.quality

print("Xv:", Xv.shape)
print("yv:", yv.shape)

# Complete data
complete = pd.concat([wine_test, wine_train])
X = complete.drop(['quality'], axis=1)
y = complete.quality

print('X:', X.shape)
print('y:', y.shape)
```

```
Xt: (3000, 12)
yt: (3000,)
Xv: (1000, 12)
yv: (1000,)
X: (4000, 12)
y: (4000,)
```

[12]:
```
def ovr_roc_plot(y_true, y_pred):
    """ Draw ROC curves using one-vs-rest approach
    """

    classes = y_true.unique()
    n_classes = len(y_true.unique())

    # Convert from n x 1 categorical matrix to n x k binary matrix
    y_true = sklearn.preprocessing.label_binarize(y_true, classes)

    # Sanity Check
    if y_true.shape[1] != y_pred.shape[1]:
        raise ValueError("Truth and prediction dimensions do not match.")

    # Compute ROC curve and ROC area for each class
    rocs = dict()
```

```
    aucs = dict()
    for name, i in zip(classes, range(n_classes)):
        aucs[i] = pd.DataFrame({
            'quality': [name],
            'auc': [sklearn.metrics.roc_auc_score(y_true[:, i], y_pred[:, i])]
        })

        rocs[i] = pd.DataFrame(
            data = np.c_[sklearn.metrics.roc_curve(y_true[:, i], y_pred[:, i])],
            columns = ('fpr', 'tpr', 'threshold')
        ).assign(
            quality = name
        )

    # Bind rows to create a single data frame for each
    roc = pd.concat(rocs)
    auc = pd.concat(aucs)

    # Create plot
    sns.lineplot(x='fpr', y='tpr', hue='quality', data=roc, ci=None)

    plt.plot([0,1],[0,1], 'k--', alpha=0.5) # 0-1 line
    plt.title("ROC ovr curves")

    plt.show()

    # Return the AUCs as a Data Frame
    return(auc)
```

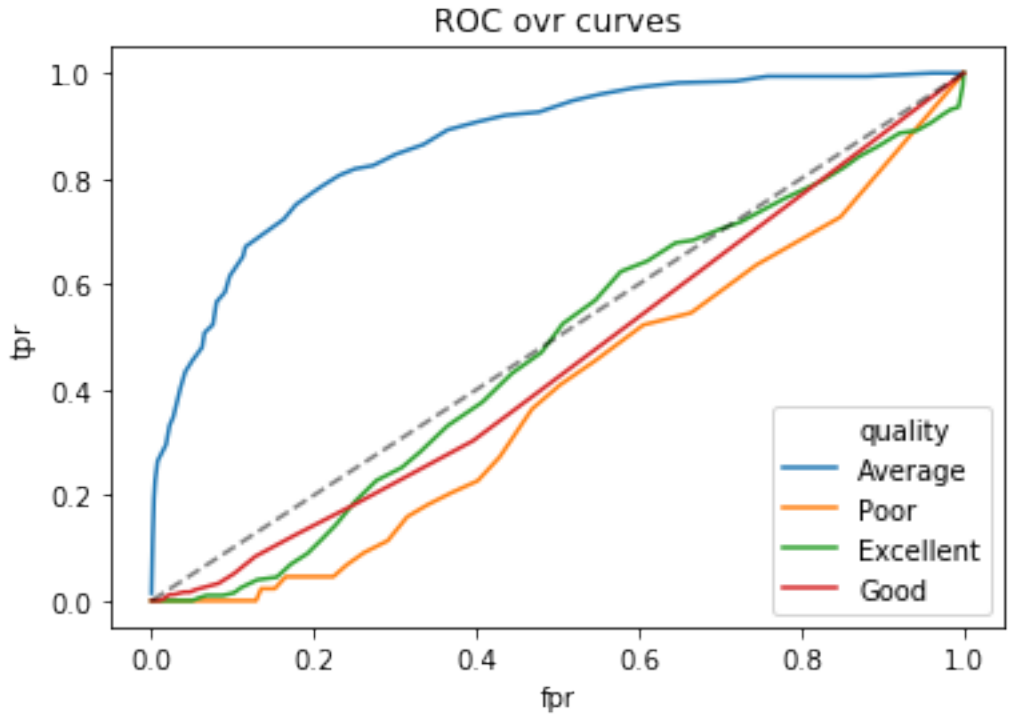
[13]:
```
def model_test_assess(model):
    # Use the model to predict test labels
    res = wine_test.copy()
    res["pred_label"] = model.predict(wine_test.drop(['quality'], axis=1))

    # Print report
    print(
        sklearn.metrics.classification_report(res.quality, res.pred_label)
    )
```

[14]:
```
qualities = ['Poor', 'Average', 'Good', 'Excellent']
```

[15]:
```
m_rf = sklearn.ensemble.RandomForestClassifier(
        n_estimators=50, max_depth=36
    ).fit(Xt,yt)
```

[16]:
```
rf = wine_test.copy()
rf['pred_label'] = m_rf.predict(rf.drop(['quality'], axis=1))
```

```
ovr_roc_plot(rf.quality, m_rf.predict_proba(rf.drop(['quality', 'pred_label'],␣
↪axis=1)))
```



[16]:

	quality	auc
0 0	Average	0.867911
1 0	Poor	0.390322
2 0	Excellent	0.469875
3 0	Good	0.449012

1.0.3 Figure 5: ROC Curves and AUC table

1.1 Other Models That Were Tested

1.1.1 Nearest Neighbors Model

[17]:
```
param = {'n_neighbors':(1,31)}
m_mn = GridSearchCV(
    sklearn.neighbors.KNeighborsClassifier(),
    param_grid=param,
    cv=5,).fit(Xt,yt)
```

```
print( "best n_neighbors:", m_mn.best_params_['n_neighbors'])
```

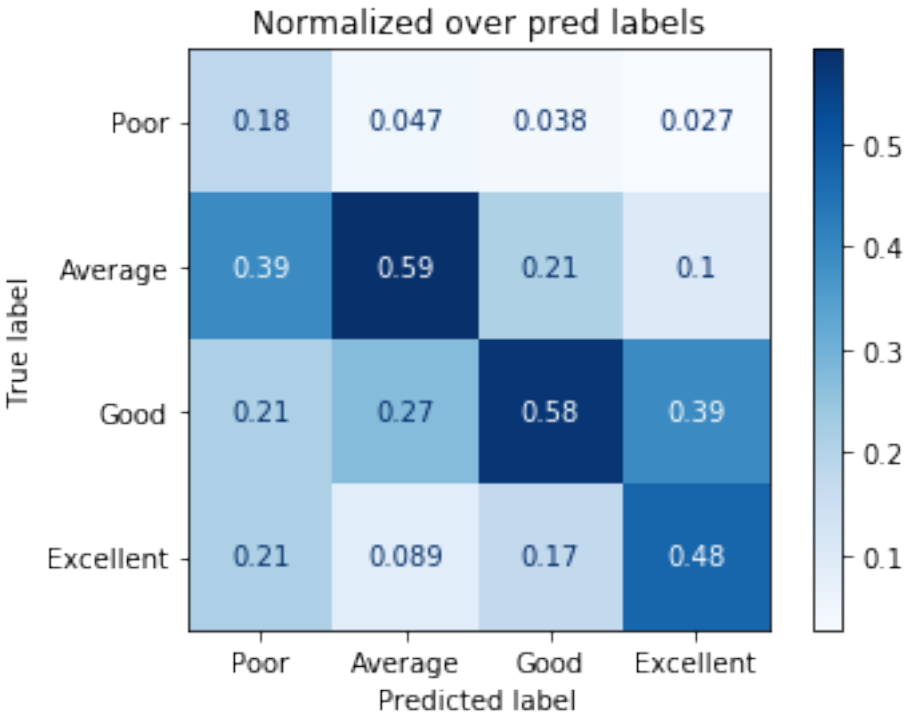
```
best n_neighbors: 1
```

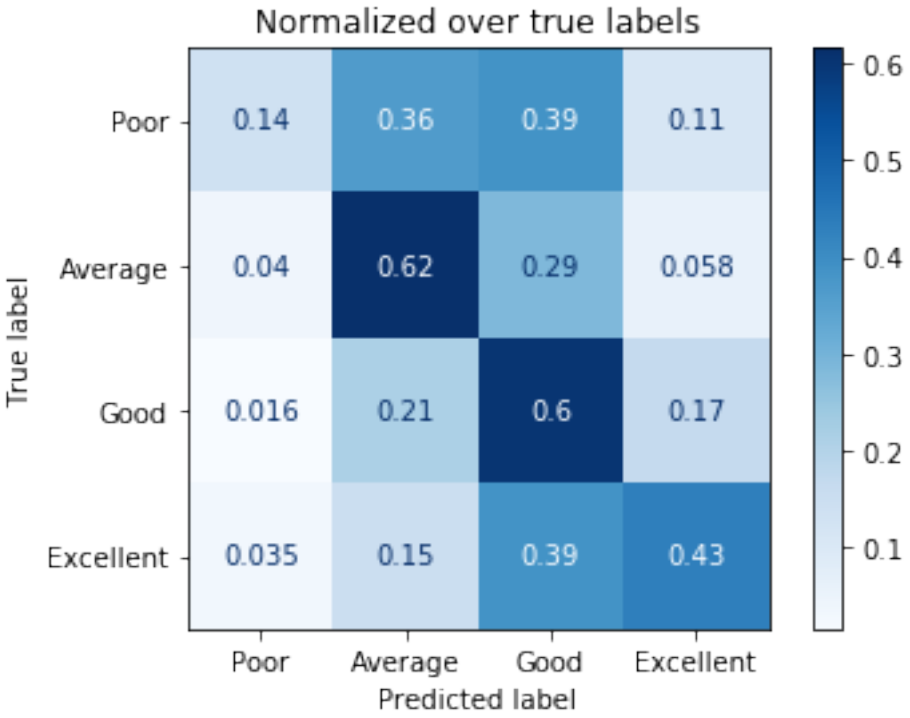
[18]:
```
@interact(n_neighbors = np.arange(1, 31))

def fit_nn(n_neighbors):
    m_nn = sklearn.neighbors.KNeighborsClassifier(n_neighbors=n_neighbors).
 ↪fit(Xt,yt)

    res_nn = wine_test.copy()
    res_nn["pred_label"] = m_nn.predict(wine_test.drop(['quality'], axis=1))

    print(sklearn.metrics.classification_report(res_nn.quality, res_nn.
 ↪pred_label)
    )
```

```
interactive(children=(Dropdown(description='n_neighbors', options=(1, 2, 3, 4, 5, 6, 7, 8, 9, 1
```

[36]:
```
m_nn = sklearn.neighbors.KNeighborsClassifier(n_neighbors=1).fit(Xt,yt)
# Plot non-normalized confusion matrix
titles_options = [("Normalized over pred labels", 'pred'),
                  ("Normalized over true labels", 'true')]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(m_nn, Xv, yv,
                                 labels=qualities,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

plt.show()
```

```
Normalized over pred labels
[[0.18181818 0.04733728 0.03811659 0.0273224 ]
 [0.39393939 0.59171598 0.20852018 0.10382514]
 [0.21212121 0.27218935 0.57847534 0.39344262]
 [0.21212121 0.0887574  0.17488789 0.47540984]]
Normalized over true labels
[[0.13636364 0.36363636 0.38636364 0.11363636]
 [0.04       0.61538462 0.28615385 0.05846154]
 [0.01631702 0.21445221 0.6013986  0.16783217]
 [0.03465347 0.14851485 0.38613861 0.43069307]]
```


Normalized over pred labels
Poor
Average
Good
Excellent
True label
0.18
0.047
0.038
0.027
0.39
0.59
0.21
0.1
0.21
0.27
0.58
0.39
0.21
0.089
0.17
0.48
Poor
Average
Good
Excellent
Predicted label
0.5
0.4
0.3
0.2
0.1


Normalized over true labels
Poor
Average
Good
Excellent
True label
0.14
0.36
0.39
0.11
0.04
0.62
0.29
0.058
0.016
0.21
0.6
0.17
0.035
0.15
0.39
0.43
Poor
Average
Good
Excellent
Predicted label
0.6
0.5
0.4
0.3
0.2
0.1

I didn't pick Nearest Neighbors Model because overall for a large range of n neighbors, the accuracy was always below 50%.

1.1.2 Decision Tree Classifier

[21]:
```
param = {'max_depth':(1,20)}
m_dt = GridSearchCV(
    sklearn.tree.DecisionTreeClassifier(),
    param_grid=param,
    cv=5,).fit(Xt,yt)

print( "best max_depth:", m_dt.best_params_['max_depth'])
```

```
best max_depth: 20
```

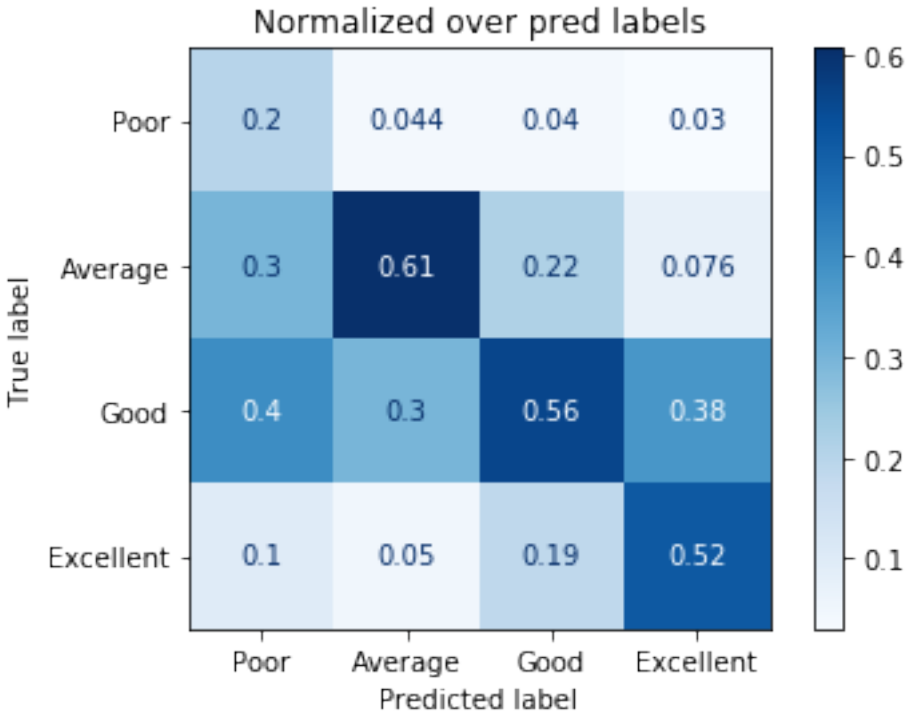
[22]:
```
@interact(max_depth = (1,20))

def fit_dt(max_depth):
    m_dt = sklearn.tree.DecisionTreeClassifier(max_depth = max_depth).fit(Xt,yt)

    res = wine_test.copy()
    res["pred_label"] = m_dt.predict(wine_test.drop(['quality'], axis=1))

    print(
        sklearn.metrics.classification_report(res.quality, res.pred_label)
    )
```

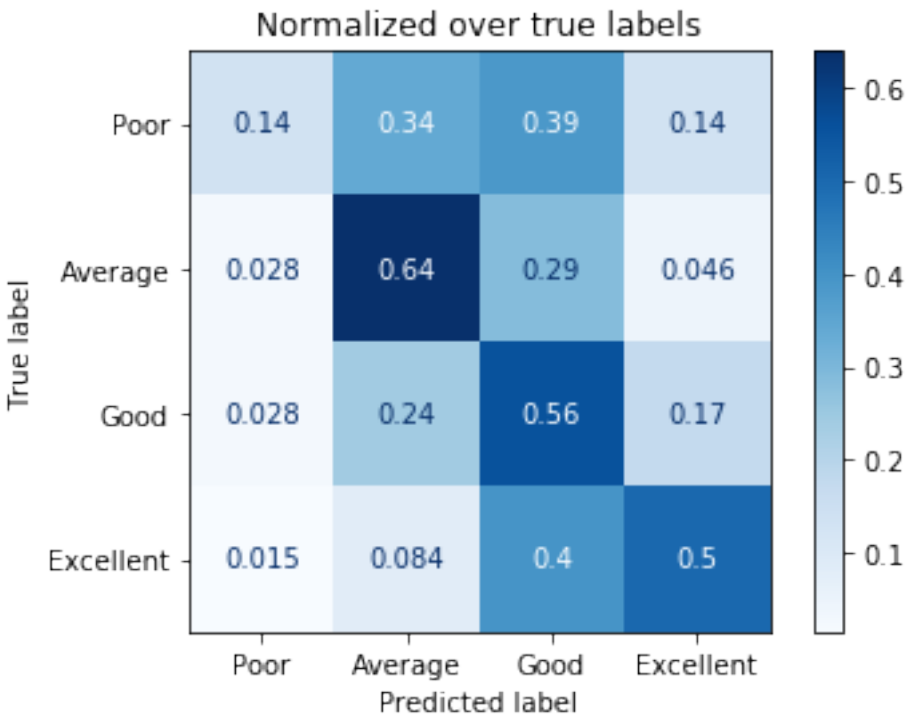
```
interactive(children=(IntSlider(value=10, description='max_depth', max=20, min=1), Output()),
```

[44]:
```
m_dt = sklearn.tree.DecisionTreeClassifier(max_depth = 20).fit(Xt,yt)
# Plot non-normalized confusion matrix
titles_options = [("Normalized over pred labels", 'pred'),
                  ("Normalized over true labels", 'true')]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(m_dt, Xv, yv,
                                 labels=qualities,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

plt.show()
```

```
Normalized over pred labels
```

```
[[0.2  0.04 0.04 0.03]
 [0.3  0.61 0.22 0.08]
 [0.4  0.3  0.56 0.38]
 [0.1  0.05 0.19 0.52]]
Normalized over true labels
[[0.14 0.34 0.39 0.14]
 [0.03 0.64 0.29 0.05]
 [0.03 0.24 0.56 0.17]
 [0.01 0.08 0.4  0.5 ]]
```





1.1.3 Logistic Regression (ovr)

If the option chosen is 'ovr', then a binary problem is fit for each label.

[51]:
```
m_ovr = make_pipeline(
        sklearn.preprocessing.StandardScaler(),
        LogisticRegression(penalty='none', multi_class='ovr')
    ).fit(Xt,yt)
```

[52]:
```
ovr = wine_test.copy()
ovr['pred_label'] = m_ovr.predict(ovr.drop(['quality'], axis=1))
```

[53]:
```
model_test_assess(m_ovr)
```

```
              precision    recall  f1-score   support

     Average       0.59      0.66      0.62       325
   Excellent       0.59      0.16      0.26       202
        Good       0.51      0.69      0.59       429
        Poor       0.00      0.00      0.00        44

    accuracy                           0.54      1000
   macro avg       0.42      0.38      0.37      1000
```

```
weighted avg       0.53      0.54      0.51      1000
```

[54]:
```
pred_probs = pd.DataFrame(
    # We round here for the sake of nicer printing in the table below
    data = np.round(m_ovr.predict_proba(ovr.drop(['quality', 'pred_label'],␣
 ↪axis=1)),4),
    columns = qualities
)
# Insert the predicted labels
pred_probs.insert(
    0, 'pred_label', ovr.pred_label
)

pred_probs
```

[54]:

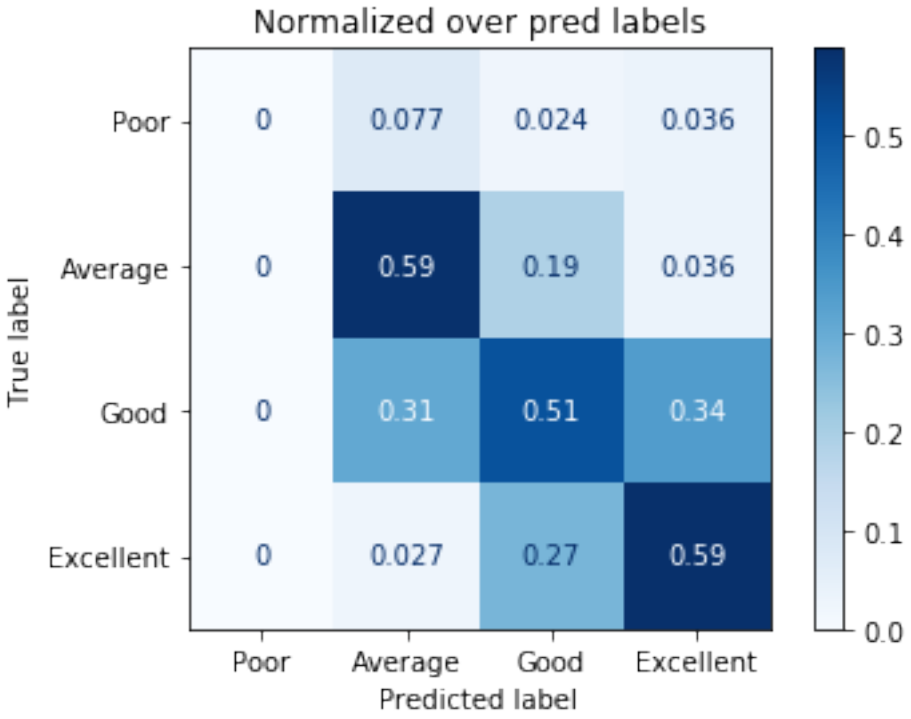
	pred_label	Poor	Average	Good	Excellent
0	Average	0.5969	0.0269	0.3663	0.0098
1	Average	0.6335	0.0200	0.2975	0.0490
2	Good	0.0976	0.3484	0.5481	0.0060
3	Average	0.4664	0.0364	0.4262	0.0710
4	Average	0.6222	0.0247	0.2673	0.0858
..	…	…	…	…	…
995	Good	0.0580	0.4144	0.5230	0.0047
996	Average	0.5431	0.0805	0.3238	0.0526
997	Average	0.6092	0.0258	0.2752	0.0898
998	Average	0.5023	0.0479	0.3248	0.1250
999	Good	0.1564	0.3653	0.4695	0.0088

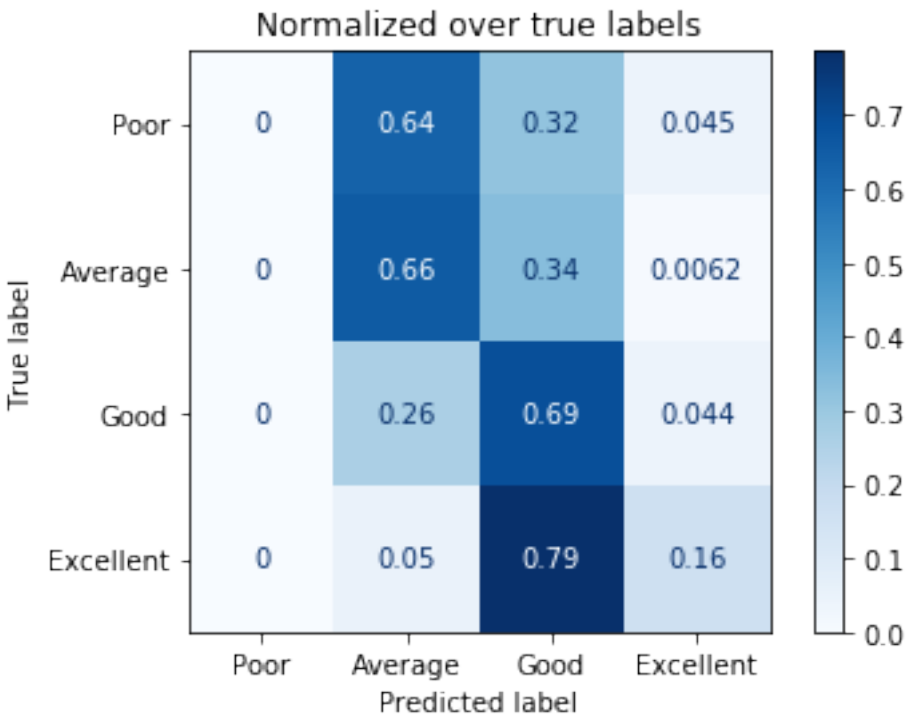
[1000 rows x 5 columns]

[55]:
```
# Plot non-normalized confusion matrix
titles_options = [("Normalized over pred labels", 'pred'),
                  ("Normalized over true labels", 'true')]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(m_ovr, Xv, yv,
                                 labels=qualities,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

plt.show()
```

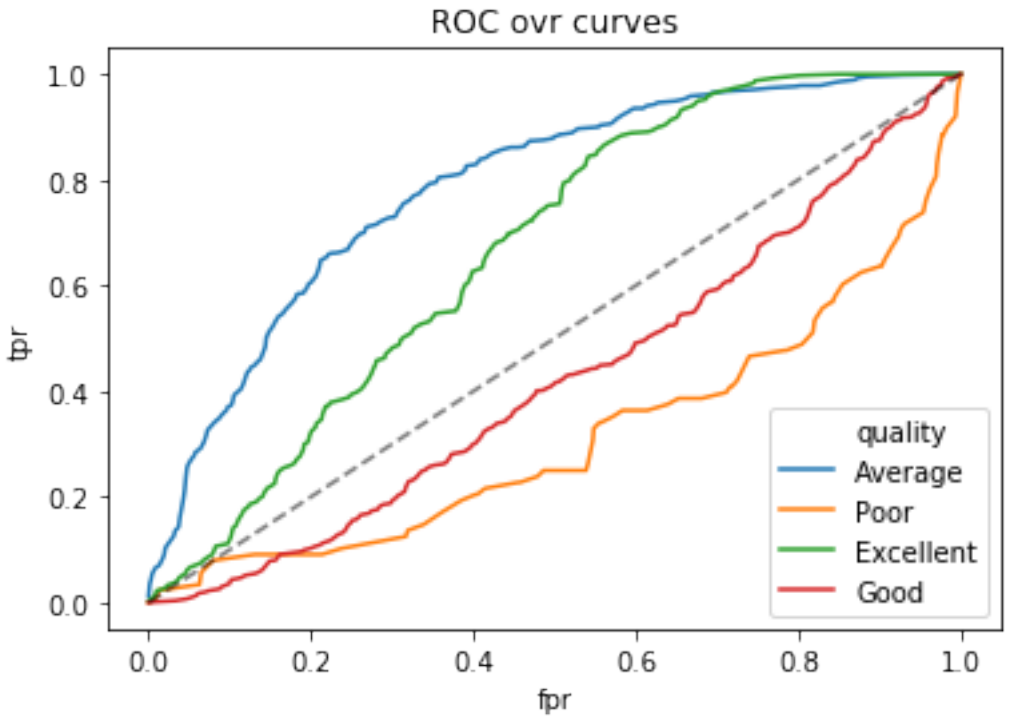
```
Normalized over pred labels
[[0.   0.08 0.02 0.04]
```

```
 [0.   0.59 0.19 0.04]
 [0.   0.31 0.51 0.34]
 [0.   0.03 0.27 0.59]]
Normalized over true labels
[[0.   0.64 0.32 0.05]
 [0.   0.66 0.34 0.01]
 [0.   0.26 0.69 0.04]
 [0.   0.05 0.79 0.16]]
```





[29]:
```
ovr_roc_plot(ovr.quality, m_ovr.predict_proba(ovr.drop(['quality',␣
↪'pred_label'], axis=1)))
```



[29]:

		quality	auc
0	0	Average	0.786379
1	0	Poor	0.311359
2	0	Excellent	0.667287
3	0	Good	0.423018

1.1.4 Logisistic Regression (multinomial)

For 'multinomial' the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary.

[30]:
```
m_mn = LogisticRegression(penalty='none', multi_class='multinomial',␣
↪max_iter=1000).fit(Xt,yt)
```

[31]:
```
model_test_assess(m_mn)
```

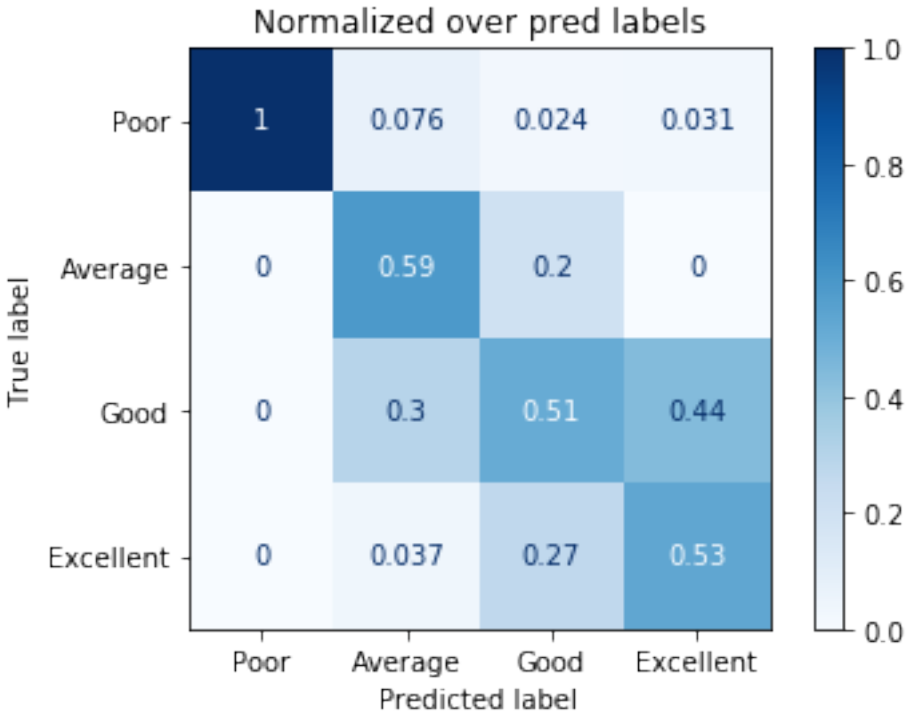
```
              precision    recall  f1-score   support

     Average       0.59      0.64      0.62       325
   Excellent       0.53      0.17      0.26       202
        Good       0.51      0.69      0.59       429
        Poor       1.00      0.02      0.04        44

    accuracy                           0.54      1000
   macro avg       0.66      0.38      0.38      1000
weighted avg       0.56      0.54      0.51      1000
```

[32]:
```
mn = wine_test.copy()
mn['pred_label'] = m_mn.predict(mn.drop(['quality'], axis=1))
```

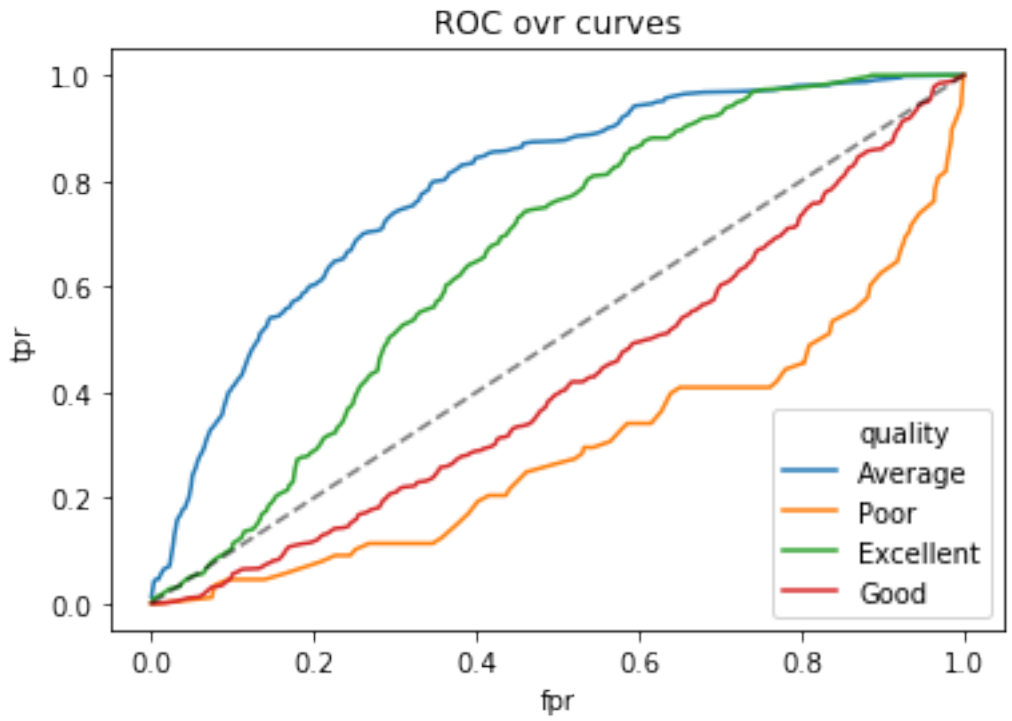
[33]:
```
pred_probs = pd.DataFrame(
    # We round here for the sake of nicer printing in the table below
    data = np.round(m_mn.predict_proba(mn.drop(['quality', 'pred_label'],␣
↪axis=1)),4),
    columns = qualities
)
# Insert the predicted labels
pred_probs.insert(
    0, 'pred_label', mn.pred_label
)

# pred_probs
```

[42]:
```
# Plot non-normalized confusion matrix
titles_options = [("Normalized over pred labels", 'pred'),
                  ("Normalized over true labels", 'true')]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(m_mn, Xv, yv,
                                 labels=qualities,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

plt.show()
```

```
Normalized over pred labels
[[1.   0.08 0.02 0.03]
 [0.   0.59 0.2  0.  ]
 [0.   0.3  0.51 0.44]
 [0.   0.04 0.27 0.53]]
Normalized over true labels
[[0.02 0.61 0.32 0.05]
 [0.   0.64 0.36 0.  ]
 [0.   0.24 0.69 0.07]
 [0.   0.06 0.77 0.17]]
```



[26]:
```
ovr_roc_plot(mn.quality, m_mn.predict_proba(mn.drop(['quality', 'pred_label'],␣
↪axis=1)))
```



[26]:

	quality	auc
0 0	Average	0.789356
1 0	Poor	0.293553
2 0	Excellent	0.659216
3 0	Good	0.423614

1.1.5 Support Vector Machines

[39]:
```
param = {'C':np.logspace(-2,3,6), 'kernel':['rbf']}
m_dt = GridSearchCV(
    sklearn.svm.SVC(),
    param_grid=param,
    cv=5,).fit(Xt,yt)

print( "best C:", m_dt.best_params_['C'])
print( "best kernel:", m_dt.best_params_['kernel'])
```

```
best C: 1000.0
best kernel: rbf
```

[60]:
```
# param = {'C':np.logspace(-2,3,6), 'kernel':['linear']}
# m_dt = GridSearchCV(
```

```
#     sklearn.svm.SVC(),
#     param_grid=param,
#     cv=5,).fit(Xt,yt)

# print( "best C:", m_dt.best_params_['C'])
# print( "best kernel:", m_dt.best_params_['kernel'])
```

[59]:
```
# param = {'C':np.logspace(-2,3,6), 'kernel':['poly']}
# m_dt = GridSearchCV(
#     sklearn.svm.SVC(),
#     param_grid=param,
#     cv=5,).fit(Xt,yt)

# print( "best C:", m_dt.best_params_['C'])
# print( "best kernel:", m_dt.best_params_['kernel'])
```

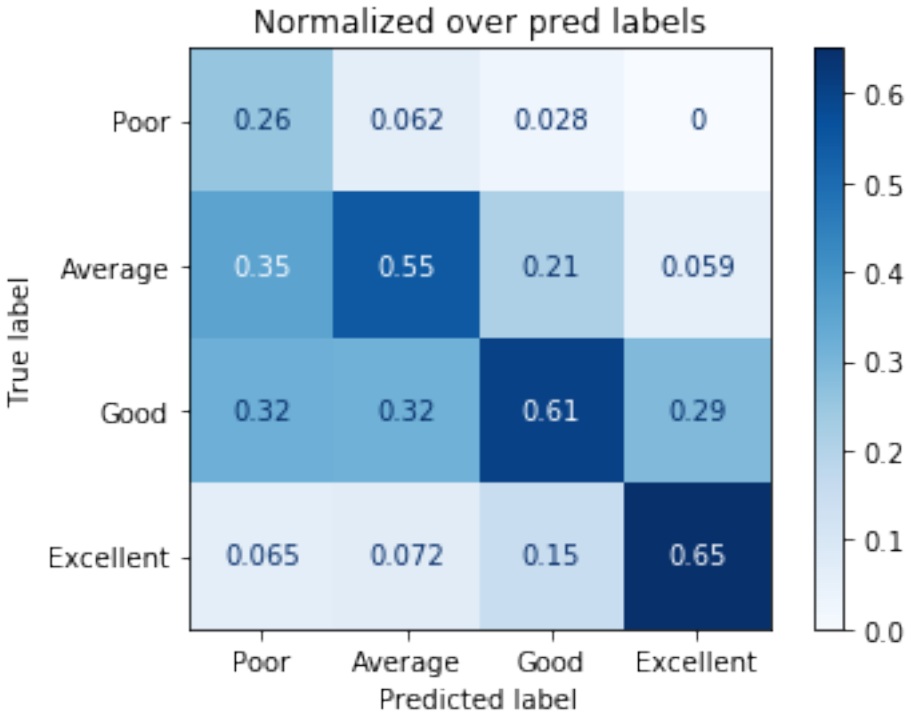
[58]:
```
@interact(C = np.logspace(-2,3,6), kernel = ['rbf', 'linear', 'poly'])

def fit_svc(C=1, kernel='rbf'):
    m_svc = make_pipeline(
        sklearn.preprocessing.StandardScaler(),
        sklearn.svm.SVC(C=C, kernel=kernel)
    ).fit(Xt,yt)

    model_test_assess(m_svc)
```

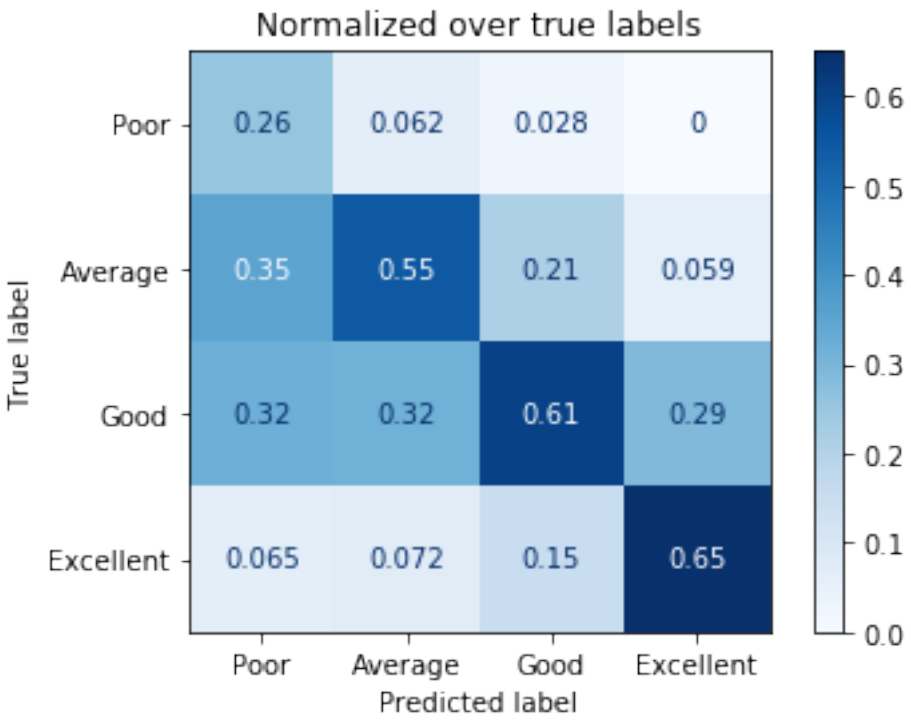
```
interactive(children=(Dropdown(description='C', index=2, options=(0.01, 0.1, 1.0, 10.0, 100.0,
```

[45]:
```
classifier = make_pipeline(
        sklearn.preprocessing.StandardScaler(),
        sklearn.svm.SVC(C=1000, kernel='rbf')
    ).fit(Xt,yt)

np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
titles_options = [("Normalized over pred labels", 'pred'),
                  ("Normalized over true labels", 'true')]

for title, normalize in titles_options:
    disp = plot_confusion_matrix(classifier, Xv, yv,
                                 labels=qualities,
                                 cmap=plt.cm.Blues,
                                 normalize='pred')
    disp.ax_.set_title(title)
```

```
    print(title)
    print(disp.confusion_matrix)

plt.show()
```

```
Normalized over pred labels
[[0.26 0.06 0.03 0.  ]
 [0.35 0.55 0.21 0.06]
 [0.32 0.32 0.61 0.29]
 [0.06 0.07 0.15 0.65]]
Normalized over true labels
[[0.26 0.06 0.03 0.  ]
 [0.35 0.55 0.21 0.06]
 [0.32 0.32 0.61 0.29]
 [0.06 0.07 0.15 0.65]]
```




Normalized over true labels
Poor
Average
Good
Excellent
True label
0.26
0.062
0.028
0
0.35
0.55
0.21
0.059
0.32
0.32
0.61
0.29
0.065
0.072
0.15
0.65
Poor
Average
Good
Excellent
Predicted label
0.6
0.5
0.4
0.3
0.2
0.1
0.0

[ ]: