Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Отчет

по дисциплине «Технология Машинного обучения»

Выполнил:
студент группы ИУ5-62
Миронов Святослав
подпись, дата

2019 г.

Задание:

1. Выберите набор данных (датасет) для решения задачи классификации или регресии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
4. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью трех подходящих для задачи метрик.
5. Постройте модель и оцените качество модели с использованием кросс-валидации. Проведите эксперименты с тремя различными стратегиями кросс-валидации.
6. Произведите подбор гиперпараметра K с использованием GridSearchCV и кросс-валидации.
7. Повторите пункт 4 для найденного оптимального значения гиперпараметра K. Сравните качество полученной модели с качеством модели, полученной в пункте 4.
8. Постройте кривые обучения и валидации.

Lab4_end

June 3, 2019

In [1]:
```
!pip install seaborn
!pip install lightgbm
```

```
Requirement already satisfied: seaborn in /srv/conda/lib/python3.6/site-packages (0.9.0)
Requirement already satisfied: scipy>=0.14.0 in /srv/conda/lib/python3.6/site-packages (from se
Requirement already satisfied: matplotlib>=1.4.3 in /srv/conda/lib/python3.6/site-packages (fro
Requirement already satisfied: pandas>=0.15.2 in /srv/conda/lib/python3.6/site-packages (from s
Requirement already satisfied: numpy>=1.9.3 in /srv/conda/lib/python3.6/site-packages (from sea
Requirement already satisfied: cycler>=0.10 in /srv/conda/lib/python3.6/site-packages (from mat
Requirement already satisfied: kiwisolver>=1.0.1 in /srv/conda/lib/python3.6/site-packages (fro
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /srv/conda/lib/pytho
Requirement already satisfied: python-dateutil>=2.1 in /srv/conda/lib/python3.6/site-packages
Requirement already satisfied: pytz>=2011k in /srv/conda/lib/python3.6/site-packages (from pand
Requirement already satisfied: six in /srv/conda/lib/python3.6/site-packages (from cycler>=0.10
Requirement already satisfied: setuptools in /srv/conda/lib/python3.6/site-packages (from kiwis
Requirement already satisfied: lightgbm in /srv/conda/lib/python3.6/site-packages (2.2.3)
Requirement already satisfied: scipy in /srv/conda/lib/python3.6/site-packages (from lightgbm)
Requirement already satisfied: scikit-learn in /srv/conda/lib/python3.6/site-packages (from lig
Requirement already satisfied: numpy in /srv/conda/lib/python3.6/site-packages (from lightgbm)
```

In [29]:
```
import numpy as np
import pandas as pd
from sklearn import datasets
import matplotlib.pyplot as plt
import seaborn as sns
import lightgbm


from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_re
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.model_selection import learning_curve, validation_curve
```

```
%matplotlib inline
```

```
In [3]: df = pd.read_csv('heart.csv', sep=",")
```

1 Data Set Information:

This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "goal" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

1.1 Attribute Information:

1. age
2. sex
3. chest pain type (4 values)
4. resting blood pressure
5. serum cholestoral in mg/dl
6. fasting blood sugar > 120 mg/dl
7. resting electrocardiographic results (values 0,1,2)
8. maximum heart rate achieved
9. exercise induced angina
10. oldpeak = ST depression induced by exercise relative to rest
11. the slope of the peak exercise ST segment
12. number of major vessels (0-3) colored by flourosopy
13. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect

```
In [4]: df.shape
```

```
Out[4]: (303, 14)
```

```
In [5]: df.head()
```

```
Out[5]:    age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
        0   63    1   3       145   233    1        0      150      0      2.3      0
        1   37    1   2       130   250    0        1      187      0      3.5      0
        2   41    0   1       130   204    0        0      172      0      1.4      2
        3   56    1   1       120   236    0        1      178      0      0.8      2
        4   57    0   0       120   354    0        1      163      1      0.6      2

           ca  thal  target
        0   0     1       1
        1   0     2       1
        2   0     2       1
        3   0     2       1
        4   0     2       1
```

```
In [6]: df.dtypes
```

```
Out[6]: age           int64
        sex           int64
        cp            int64
        trestbps      int64
        chol          int64
        fbs           int64
        restecg       int64
        thalach       int64
        exang         int64
        oldpeak     float64
        slope         int64
        ca            int64
        thal          int64
        target        int64
        dtype: object
```

, :

```
In [7]: df.isnull().sum()

Out[7]: age         0
        sex         0
        cp          0
        trestbps    0
        chol        0
        fbs         0
        restecg     0
        thalach     0
        exang       0
        oldpeak     0
        slope       0
        ca          0
        thal        0
        target      0
        dtype: int64
```

.
:

```
In [8]: df['target'].unique()

Out[8]: array([1, 0])
```

-,

```
In [9]: #sns.pairplot(df, hue= "target")
```

In [10]:
```
plt.figure(figsize=(15, 10))
sns.heatmap(df.corr(method='spearman'),annot=True, fmt='.3f')
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f185923ac88>
```

Out[10]:

In [11]:
```
lgbm_regressor = lightgbm.LGBMRegressor().fit(df.loc[:, df.columns != 'target'], df['t

list_of_importances = list(zip(df.loc[:, df.columns != 'target'].columns.tolist(),
                               lgbm_regressor.feature_importances_))

sorted(list_of_importances, key= lambda x: x[1], reverse= True)
```

Out[11]:
```
[('chol', 200),
 ('thalach', 160),
 ('age', 142),
 ('oldpeak', 132),
 ('trestbps', 123),
 ('ca', 58),
 ('cp', 51),
 ('thal', 45),
```

```
('restecg', 39),
('sex', 38),
('slope', 34),
('exang', 23),
('fbs', 14)]
```

```
In [12]: ##important_features = [x[0] for x in sorted(list_of_importances, key= lambda x: x[1]
```

1.2 .

```
In [13]: X_train, X_test, y_train, y_test = train_test_split(df.loc[:, df.columns != 'target']
                                                             df['target'],
                                                             test_size= 0.2,
                                                             random_state= 42)
```

```
In [14]: KNeighborsClassifierObj = KNeighborsClassifier(n_neighbors=5)
         KNeighborsClassifierObj.fit(X_train, y_train)
         result_y1=KNeighborsClassifierObj.predict(X_test)
```

```
In [15]: accuracy_score(y_test, result_y1)
```

```
Out[15]: 0.6885245901639344
```

```
In [16]: from sklearn.utils.multiclass import unique_labels
         def plot_confusion_matrix(y_true, y_pred, classes,
                                   normalize=False,
                                   title=None,
                                   cmap=plt.cm.Blues):
             """
             This function prints and plots the confusion matrix.
             Normalization can be applied by setting `normalize=True`.
             """
             if not title:
                 if normalize:
                     title = 'Normalized confusion matrix'
                 else:
                     title = 'Confusion matrix, without normalization'

             # Compute confusion matrix
             cm = confusion_matrix(y_true, y_pred)
             # Only use the labels that appear in the data
             classes = classes[unique_labels(y_true, y_pred)]
             if normalize:
                 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
                 print("Normalized confusion matrix")
             else:
                 print('Confusion matrix, without normalization')

             fig, ax = plt.subplots()
```

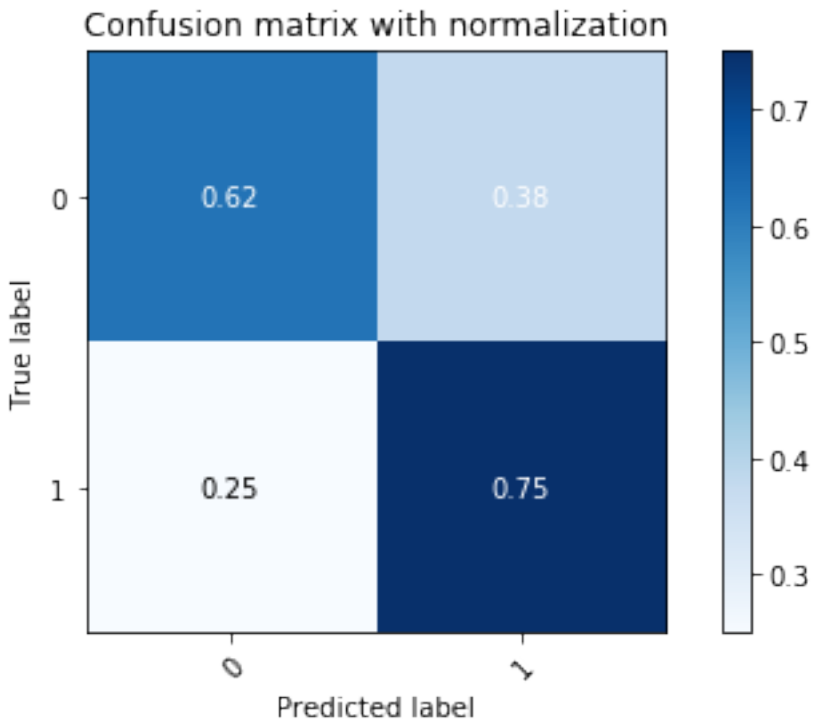
```
            im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
            ax.figure.colorbar(im, ax=ax)
            # We want to show all ticks...
            ax.set(xticks=np.arange(cm.shape[1]),
                   yticks=np.arange(cm.shape[0]),
                   # ... and label them with the respective list entries
                   xticklabels=classes, yticklabels=classes,
                   title=title,
                   ylabel='True label',
                   xlabel='Predicted label')

            # Rotate the tick labels and set their alignment.
            plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
                     rotation_mode="anchor")

            # Loop over data dimensions and create text annotations.
            fmt = '.2f' if normalize else 'd'
            thresh = cm.max() / 2.
            for i in range(cm.shape[0]):
                for j in range(cm.shape[1]):
                    ax.text(j, i, format(cm[i, j], fmt),
                            ha="center", va="center",
                            color="white" if cm[i, j] > thresh else "black")
            fig.tight_layout()
            return ax
```

```
In [17]: plot_confusion_matrix(y_test, result_y1,
                               classes=np.array(['0', '1']),
                               normalize=True,
                               title='Confusion matrix with normalization')
```

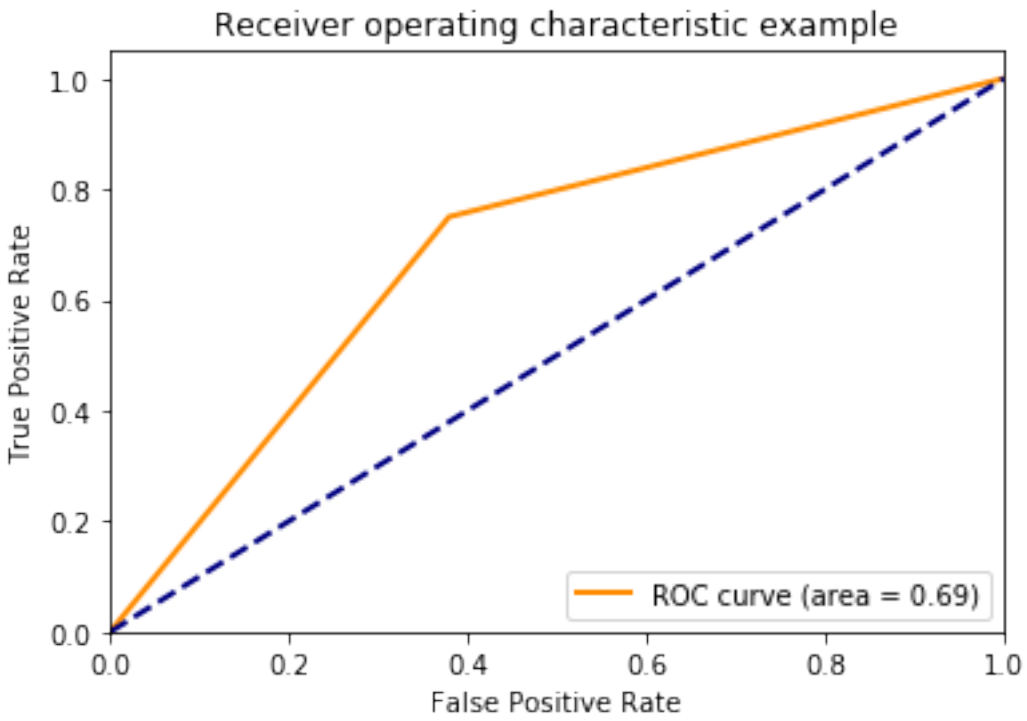
```
Normalized confusion matrix
```

```
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1855485240>
```

Out[17]:




```
In [18]: #  ROC-
         def draw_roc_curve(y_true, y_score, pos_label, average):
             fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                              pos_label=pos_label)
             roc_auc_value = roc_auc_score(y_true, y_score, average=average)
             plt.figure()
             lw = 2
             plt.plot(fpr, tpr, color='darkorange',
                      lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
             plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
             plt.xlim([0.0, 1.0])
             plt.ylim([0.0, 1.05])
             plt.xlabel('False Positive Rate')
             plt.ylabel('True Positive Rate')
             plt.title('Receiver operating characteristic example')
             plt.legend(loc="lower right")
             plt.show()

In [19]: draw_roc_curve(y_test, result_y1, pos_label=1, average='micro')

Out[19]:
```




```
In [20]: precision_score(y_test, result_y1), recall_score(y_test, result_y1)

Out[20]: (0.6857142857142857, 0.75)

In [21]: random_search = GridSearchCV(estimator= KNeighborsClassifier(),
                                      param_grid= {'n_neighbors': [5,10,12,13,14,15,16,20,30,5
                                      scoring= 'f1_weighted',
                                      cv= 3)

         random_search.fit(df.loc[:, df.columns != 'target'], df['target'])

Out[21]: GridSearchCV(cv=3, error_score='raise-deprecating',
                estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowsk
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform'),
                fit_params=None, iid='warn', n_jobs=None,
                param_grid={'n_neighbors': [5, 10, 12, 13, 14, 15, 16, 20, 30, 50]},
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring='f1_weighted', verbose=0)

In [22]: random_search.best_params_

Out[22]: {'n_neighbors': 13}

In [23]: random_search.best_score_
```

Out[23]: 0.6752523247180728

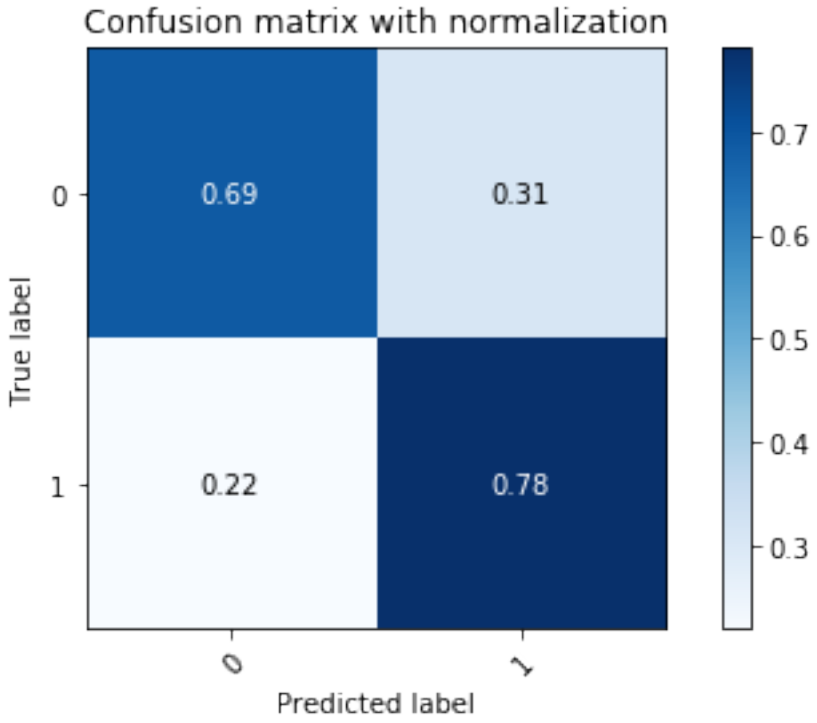
In [24]:
```
KNeighborsClassifierObj = KNeighborsClassifier(n_neighbors=13)
KNeighborsClassifierObj.fit(X_train, y_train)
result_y2=KNeighborsClassifierObj.predict(X_test)
```

In [25]:
```
plot_confusion_matrix(y_test, result_y2,
                      classes=np.array(['0', '1']),
                      normalize=True,
                      title='Confusion matrix with normalization')
```

```
Normalized confusion matrix
```

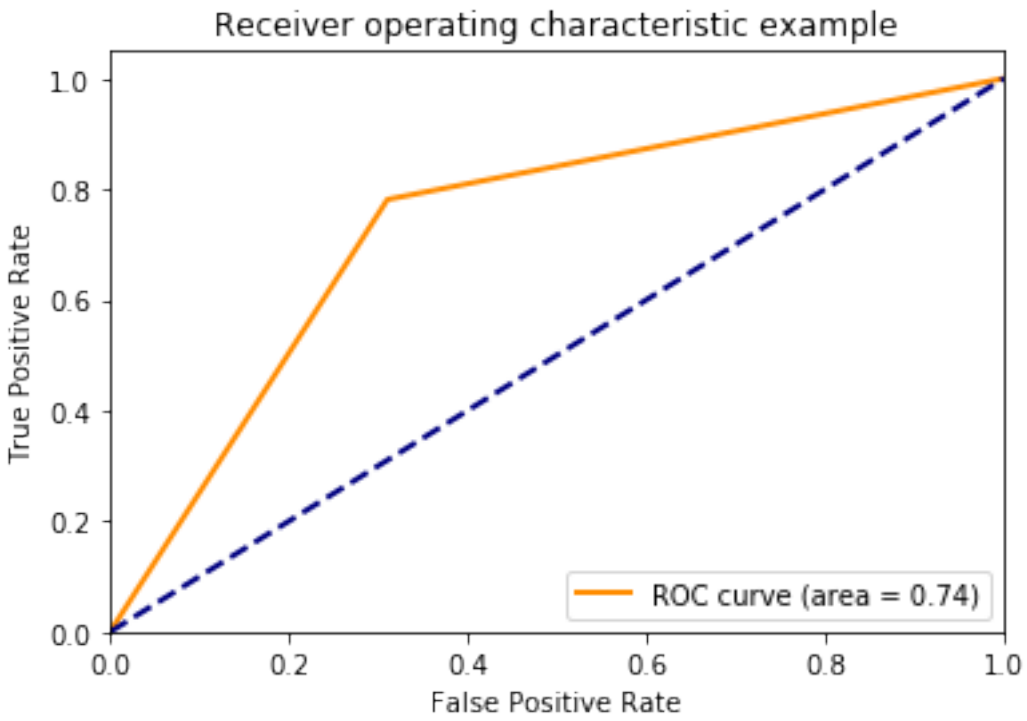
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7f185581b5f8>

Out[25]:



In [26]:
```
draw_roc_curve(y_test, result_y2, pos_label=1, average='micro')
```

Out[26]:




```
In [27]: def plot_validation_curve(estimator, title, X, y,
                                   param_name, param_range, cv,
                                   scoring="accuracy"):

            train_scores, test_scores = validation_curve(
                estimator, X, y, param_name=param_name, param_range=param_range,
                cv=cv, scoring=scoring, n_jobs=1)
            train_scores_mean = np.mean(train_scores, axis=1)
            train_scores_std = np.std(train_scores, axis=1)
            test_scores_mean = np.mean(test_scores, axis=1)
            test_scores_std = np.std(test_scores, axis=1)

            plt.title(title)
            plt.xlabel(param_name)
            plt.ylabel("Score")
            plt.ylim(0.0, 1.1)
            lw = 2
            plt.plot(param_range, train_scores_mean, label="Training score",
                         color="darkorange", lw=lw)
            plt.fill_between(param_range, train_scores_mean - train_scores_std,
                             train_scores_mean + train_scores_std, alpha=0.2,
                             color="darkorange", lw=lw)
            plt.plot(param_range, test_scores_mean, label="Cross-validation score",
                         color="navy", lw=lw)
```

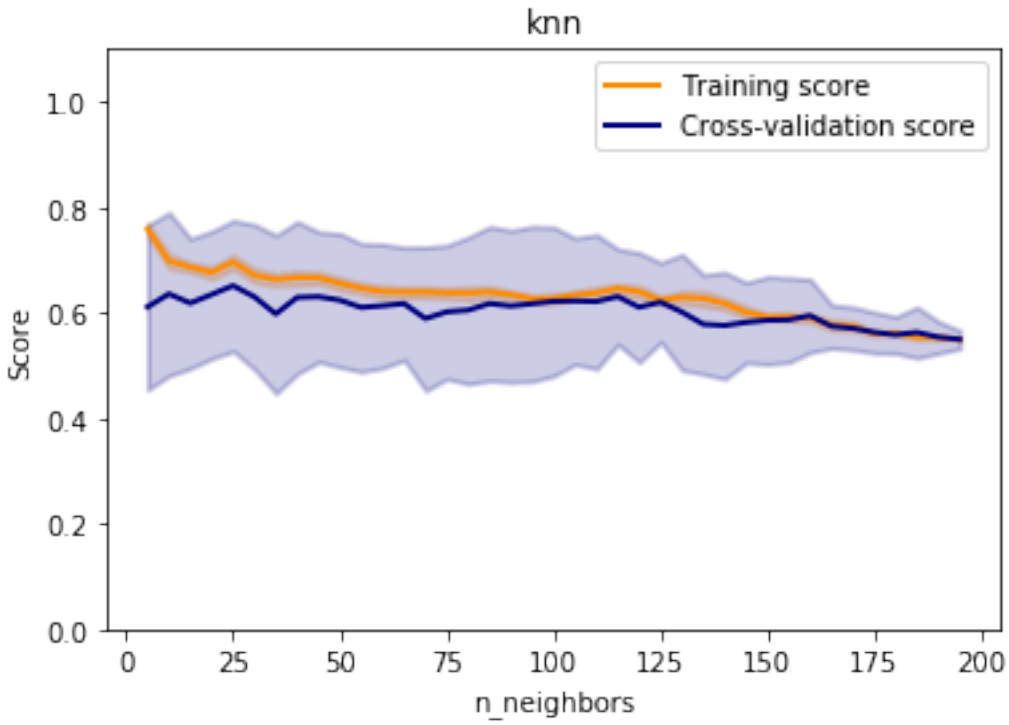
```
            plt.fill_between(param_range, test_scores_mean - test_scores_std,
                             test_scores_mean + test_scores_std, alpha=0.2,
                             color="navy", lw=lw)
            plt.legend(loc="best")
            return plt
```

In [31]:
```
n_range2 = np.array(range(5,200,5))
```

```
plot_validation_curve(KNeighborsClassifier(), 'knn',
                      X_train, y_train,
                      param_name='n_neighbors', param_range=n_range2,
                      cv=20, scoring="accuracy")
```

Out[31]:
```
<module 'matplotlib.pyplot' from '/srv/conda/lib/python3.6/site-packages/matplotlib/py
```

Out[31]:



In [0]: