МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Кафедра «Систем обработки информации и управления»

ОТЧЕТ

Лабораторная работа №4
по курсу «Методы машинного обучения»

Тема: «Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей.»

ИСПОЛНИТЕЛЬ: Ромичева Е.В.

группа ИУ5-22М ____________________
подпись

"__"_________2019 г.

ПРЕПОДАВАТЕЛЬ: Гапанюк Ю.Е.

____
подпись

"__"_________2019 г.

Москва - 2019

Цель работы

Цель лабораторной работы: изучение сложных способов подготовки выборки и подбора гиперпараметров на примере метода ближайших соседей.

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

March 24, 2019

```
In [1]: # https://www.kaggle.com/uciml/pima-indians-diabetes-database
        # https://www.kaggle.com/amolbhivarkar/knn-for-classification-using-scikit-learn
```

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.model_selection import GridSearchCV
        from sklearn.model_selection import learning_curve, validation_curve
        from sklearn.model_selection import KFold, RepeatedKFold, LeaveOneOut, LeavePOut, Shuf
        from sklearn.model_selection import cross_val_score, cross_validate
        from sklearn.metrics import roc_curve,confusion_matrix, roc_auc_score, accuracy_score,
        plt.style.use('ggplot')
```

```
In [2]: #Load the dataset
        df = pd.read_csv('diabetes.csv')

        #Print the first 5 rows of the dataframe.
        df.head()
```

```
Out[2]:    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
        0            6      148             72             35        0  33.6
        1            1       85             66             29        0  26.6
        2            8      183             64              0        0  23.3
        3            1       89             66             23       94  28.1
        4            0      137             40             35      168  43.1

           DiabetesPedigreeFunction  Age  Outcome
        0                     0.627   50        1
        1                     0.351   31        0
        2                     0.672   32        1
        3                     0.167   21        0
        4                     2.288   33        1
```

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

```
Out[6]: (768, 9)
```

```
In [7]: for col in df.columns:
            #    -
            temp_null_count = df[df[col].isnull()].shape[0]
            print('{} - {}'.format(col, temp_null_count))
```

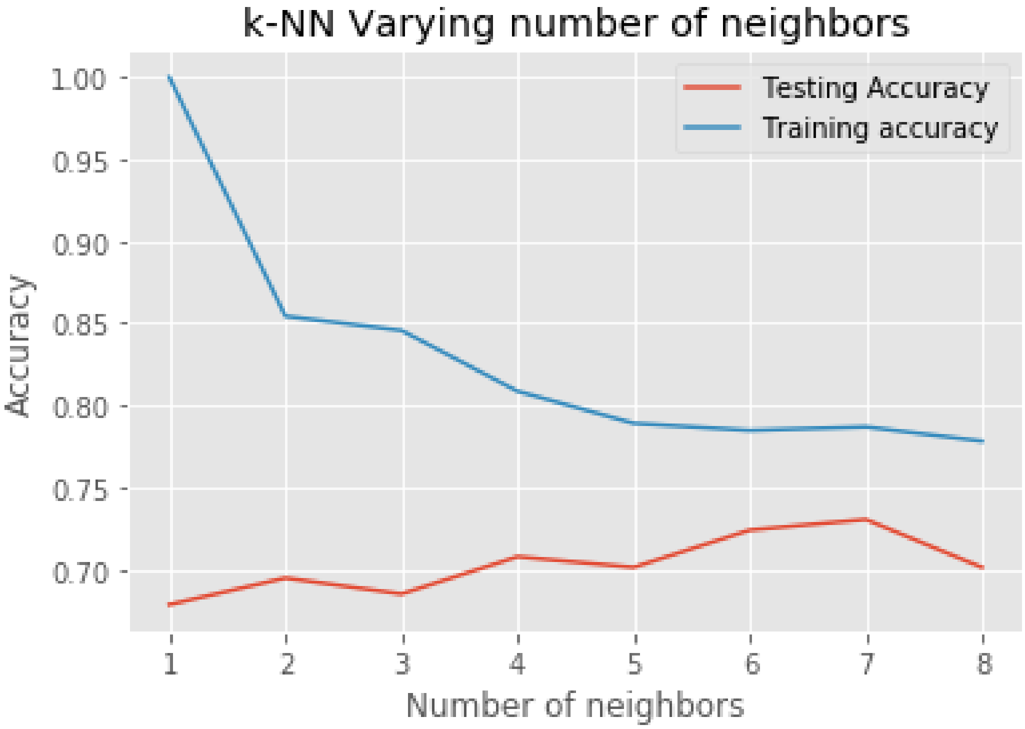
```
Pregnancies - 0
Glucose - 0
BloodPressure - 0
SkinThickness - 0
Insulin - 0
BMI - 0
DiabetesPedigreeFunction - 0
Age - 0
Outcome - 0
```

```
In [3]: X = df.drop('Outcome',axis=1).values
        y = df['Outcome'].values
```

```
In [9]: from sklearn.neighbors import KNeighborsClassifier
        from sklearn.model_selection import train_test_split
```

```
In [10]: X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.4,random_state=42, s

         #Setup arrays to store training and test accuracies
         neighbors = np.arange(1,9)
         train_accuracy =np.empty(len(neighbors))
         test_accuracy = np.empty(len(neighbors))

         for i,k in enumerate(neighbors):
             #Setup a knn classifier with k neighbors
             knn = KNeighborsClassifier(n_neighbors=k)

             #Fit the model
             knn.fit(X_train, y_train)

             #Compute accuracy on the training set
             train_accuracy[i] = knn.score(X_train, y_train)

             #Compute accuracy on the test set
             test_accuracy[i] = knn.score(X_test, y_test)
```

```
In [11]: #Generate plot
         plt.title('k-NN Varying number of neighbors')
         plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
         plt.plot(neighbors, train_accuracy, label='Training accuracy')
         plt.legend()
         plt.xlabel('Number of neighbors')
         plt.ylabel('Accuracy')
         plt.show()
```



```
In [12]: #Setup a knn classifier with k neighbors
         knn = KNeighborsClassifier(n_neighbors=7)

In [13]: #Fit the model
         knn.fit(X_train,y_train)

Out[13]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=7, p=2,
                    weights='uniform')

In [14]: #Get accuracy. Note: In case of classification algorithms score method represents acc
         knn.score(X_test,y_test)

Out[14]: 0.7305194805194806

In [16]: #import classification_report
         from sklearn.metrics import classification_report
         y_pred = knn.predict(X_test)
         print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           0       0.78      0.82      0.80       201
           1       0.62      0.56      0.59       107
```

```
   micro avg       0.73      0.73      0.73       308
   macro avg       0.70      0.69      0.70       308
weighted avg       0.73      0.73      0.73       308
```

0.0.1 Accuracy

```
In [29]: # 7
         cl1_1 = KNeighborsClassifier(n_neighbors=7)
         cl1_1.fit(X_train, y_train)
         target1_1 = cl1_1.predict(X_test)
         accuracy_score(y_test, target1_1)
```

```
Out[29]: 0.7305194805194806
```

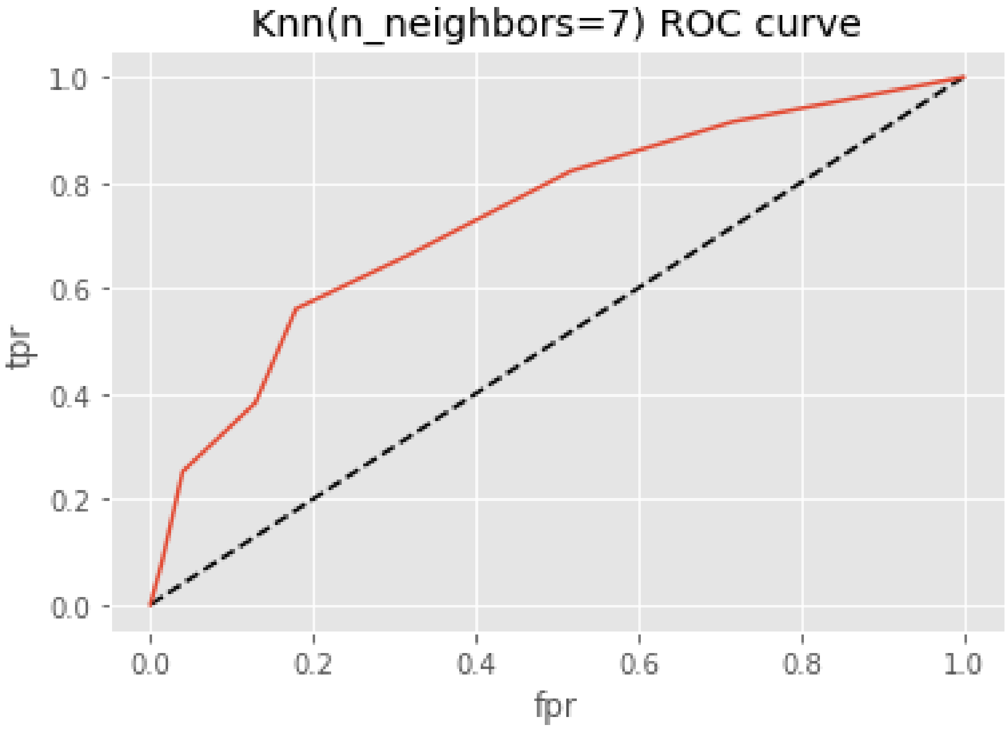
0.0.2 Confusion Matrix

```
In [21]: y_pred = knn.predict(X_test)
         confusion_matrix(y_test,y_pred)
         pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)
```

```
Out[21]: Predicted    0   1  All
         True
         0          165  36  201
         1           47  60  107
         All        212  96  308
```

0.0.3 ROC (Reciever Operating Charecteristic) curve

```
In [24]: y_pred_proba = knn.predict_proba(X_test)[:,1]
         fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
         plt.plot([0,1],[0,1],'k--')
         plt.plot(fpr,tpr, label='Knn')
         plt.xlabel('fpr')
         plt.ylabel('tpr')
         plt.title('Knn(n_neighbors=7) ROC curve')
         plt.show()
```



```
In [27]: roc_auc_score(y_test,y_pred_proba)
```

```
Out[27]: 0.7345050448691124
```

0.1 Cross Validation

```
In [31]: param_grid = {'n_neighbors':np.arange(1,50)}
         knn = KNeighborsClassifier()
         knn_cv= GridSearchCV(knn,param_grid,cv=5)
         knn_cv.fit(X,y)
```

```
Out[31]: GridSearchCV(cv=5, 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': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 1
                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])},
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring=None, verbose=0)
```

```
In [32]: knn_cv.best_score_
```

```
Out[32]: 0.7578125
```

```
In [33]: knn_cv.best_params_
```

```
Out[33]: {'n_neighbors': 14}
```

0.1.1 K-fold

```
In [38]: scores = cross_val_score(KNeighborsClassifier(n_neighbors=2),
                                  X, y,
                                  cv=KFold(n_splits=3))
         scores
```

```
Out[38]: array([0.671875  , 0.72265625, 0.73046875])
```

0.1.2 Leave One Out (LOO)

```
In [1]: # #  KFold(n_splits=n)
        # loo = LeaveOneOut()
        # loo.get_n_splits(X)

        # for train_index, test_index in loo.split(X):
        #    print("TRAIN:", train_index, "TEST:", test_index)
        #    X_train, X_test = X[train_index], X[test_index]
        #    y_train, y_test = y[train_index], y[test_index]
        #    print(X_train, X_test, y_train, y_test)
```

0.1.3 Repeated K-Fold

```
In [39]: scores2 = cross_val_score(KNeighborsClassifier(n_neighbors=2),
                                  X, y,
                                  cv=RepeatedKFold(n_splits=3, n_repeats=2))
         scores2
```

```
Out[39]: array([0.70703125, 0.70703125, 0.6953125 , 0.69140625, 0.6875    ,
                0.75      ])
```

0.2 K

```
In [41]: knn = KNeighborsClassifier(n_neighbors=14)
         knn.fit(X_train,y_train)
         knn.score(X_test,y_test)
```

```
Out[41]: 0.7305194805194806
```

```
In [42]: # accuracy
         cl1_1 = KNeighborsClassifier(n_neighbors=14)
         cl1_1.fit(X_train, y_train)
         target1_1 = cl1_1.predict(X_test)
         accuracy_score(y_test, target1_1)
```

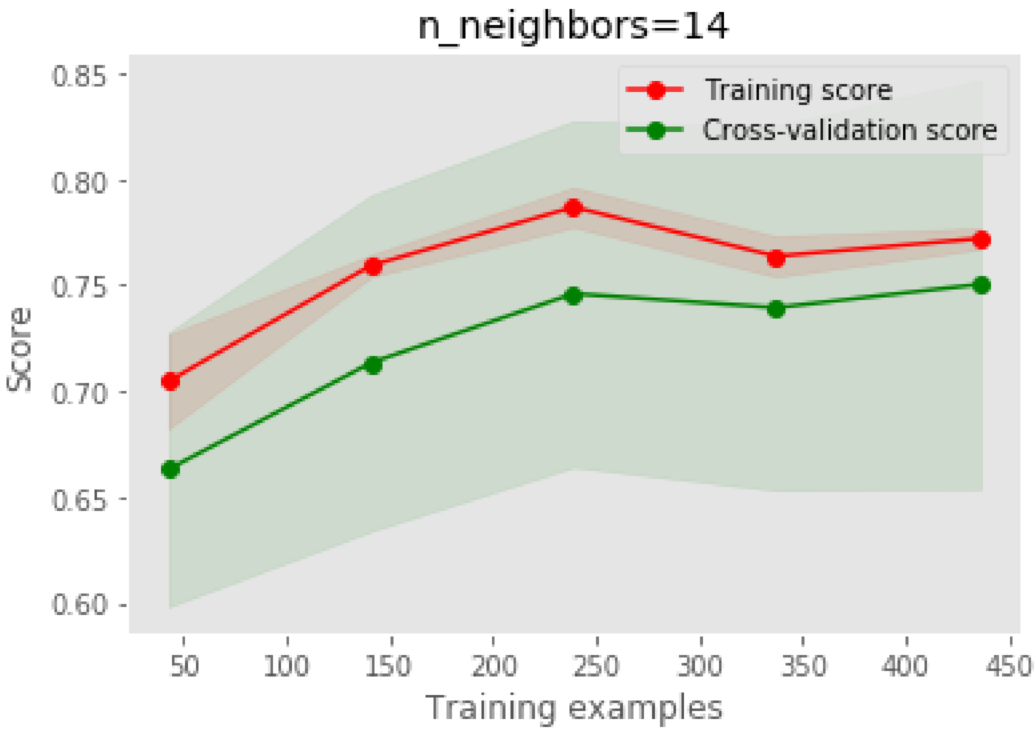
```
Out[42]: 0.7305194805194806
```

0.2.1

```
In [46]: def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                                 n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):

             plt.figure()
             plt.title(title)
             if ylim is not None:
                 plt.ylim(*ylim)
             plt.xlabel("Training examples")
             plt.ylabel("Score")
             train_sizes, train_scores, test_scores = learning_curve(
                 estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
             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.grid()

             plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                              train_scores_mean + train_scores_std, alpha=0.1,
                              color="r")
             plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                              test_scores_mean + test_scores_std, alpha=0.1, color="g")
             plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
                      label="Training score")
             plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
                      label="Cross-validation score")

             plt.legend(loc="best")
             return plt
```

```
In [47]: plot_learning_curve(KNeighborsClassifier(n_neighbors=14), 'n_neighbors=14',
                             X_train, y_train, cv=20)
```

```
Out[47]: <module 'matplotlib.pyplot' from 'c:\\users\\helen\\appdata\\local\\programs\\python\\
```

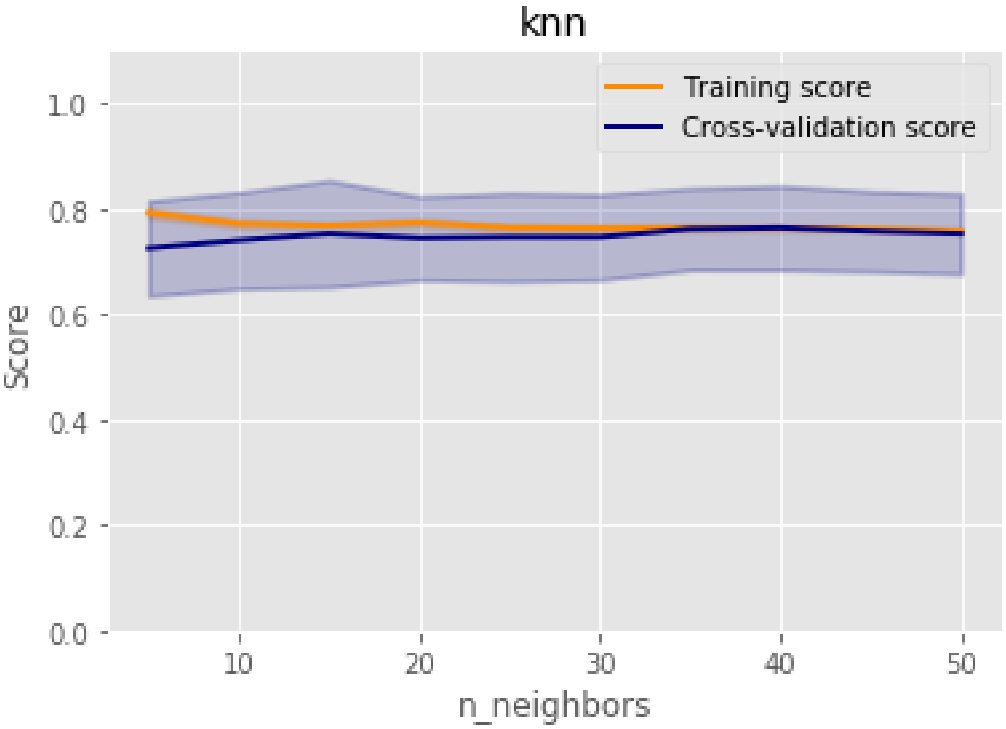


0.2.2

```
In [48]: 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 [50]: n_range = np.array(range(5,55,5))
         plot_validation_curve(KNeighborsClassifier(), 'knn',
                               X_train, y_train,
                               param_name='n_neighbors', param_range=n_range,
                               cv=20, scoring="accuracy")
```

```
Out[50]: <module 'matplotlib.pyplot' from 'c:\\users\\helen\\appdata\\local\\programs\\python\\
```



```
In [ ]:
```