Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Лабораторная работа №5
по дисциплине
«Методы машинного обучения»
на тему
«Линейные модели, SVM и деревья решений»

Выполнил:
студент группы ИУ5-24М
Мельников К. И.

Москва — 2019 г.

1. Линейные модели, SVM и деревья решений

Цель лабораторной работы: изучение линейных моделей, SVM и деревьев решений.
Задание:

```
Выберите набор данных (датасет) для решения задачи классификации или регресии.
В случае необходимости проведите удаление или заполнение пропусков и кодирование
С использованием метода train_test_split разделите выборку на обучающую и тестовую
Обучите 1) одну из линейных моделей, 2) SVM и 3) дерево решений. Оцените качество
Произведите для каждой модели подбор одного гиперпараметра с использованием GridSe
Повторите пункт 4 для найденных оптимальных значений гиперпараметров. Сравните ка
```

2. Ход работы

```
In [9]: import numpy as np

        from PIL  import Image

        from sklearn.decomposition import PCA
        from sklearn.preprocessing import scale
        from sklearn import metrics
        from sklearn.cluster import KMeans

        from IPython.display import display

        from tqdm import tqdm_notebook as tqdm

        import matplotlib.pyplot as plt
        import matplotlib.cm as cm

        from scipy import ndimage as ndi
        from skimage.morphology import medial_axis

        from scipy import ndimage as ndi
        from skimage.morphology import medial_axis
        from skimage.morphology import skeletonize

        from scipy.spatial import Delaunay

        from sklearn.model_selection import train_test_split
```

```
In [10]: data = np.load("hiragana.npz")['arr_0']
```

```
In [11]: X = []
         Y = []
         for index,letter in enumerate(data):
             for variant in letter:
                 X.append(variant)
                 Y.append(index)
```

```
In [38]: def moment(array,m1,m2):
             moment = 0
             for y,ver in enumerate(array):
                 for x,hor in enumerate(ver):
                     moment += pow(x,m1)*pow(y,m2)*hor
             return moment

         def center(array):
             x = moment(array,1,0)/moment(array,0,0)
             y = moment(array,0,1)/moment(array,0,0)
             return (x,y)

         def translate(array,x,y):
             buffer = np.roll(array,-x,axis=1)
             buffer = np.roll(buffer,-y,axis=0)
             return buffer

         def centeredarray(array):
             buffer = []
             for pic in tqdm(array):
                 shape = pic.shape
                 centroid = center(pic)
                 delta_x = -shape[1]/2 + centroid[0]
                 delta_y = -shape[0]/2 + centroid[1]

                 buffer += [translate(pic,int(delta_x),int(delta_y))]
             return buffer
```

```
In [39]: test = np.array(centeredarray(X[:640]))
```

```
HBox(children=(IntProgress(value=0, max=640), HTML(value='')))
```

```
In [137]: dataL = np.load("hirag.npz")['arr_0']
```

```
In [138]: data = np.array(dataL[:1599])
          y = np.array(Y[:1599])
```

```
In [139]: datax = data.reshape(data.shape[0], data.shape[1]*data.shape[2])
```

```
In [140]: X_train, X_test, y_train, y_test = train_test_split(datax, y, test_size
```

```
In [43]: from sklearn.linear_model import LogisticRegression
```

```
In [75]: logr = LogisticRegression(random_state=42, solver='lbfgs', multi_class='
```

```
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logisti
  "of iterations.", ConvergenceWarning)
```

```
In [76]: predicted = logr.predict(X_test)
         predictedproba = logr.predict_proba(X_test)
```

```
In [18]: from sklearn.metrics import accuracy_score
         from sklearn.metrics import roc_auc_score
         from sklearn.metrics import average_precision_score
         from sklearn.preprocessing import OneHotEncoder
```

```
In [77]: sparse = OneHotEncoder().fit_transform(y_test.reshape(-1,1)).toarray()
```

```
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/preprocessing/_encode
If you want the future behaviour and silence this warning, you can specify "catego
In case you used a LabelEncoder before this OneHotEncoder to convert the categorie
  warnings.warn(msg, FutureWarning)
```

```
In [78]: accuracy_score(y_test, predicted)
```

```
Out[78]: 0.9015151515151515
```

```
In [79]: roc_auc_score(sparse, predictedproba)
```

```
Out[79]: 0.9904642704925559
```

```
In [80]: average_precision_score(sparse, predictedproba)
```

```
Out[80]: 0.956264061746924
```

```
In [23]: from sklearn.svm import LinearSVC
```

```
In [87]: svc = LinearSVC(random_state=0, tol=1e-5).fit(X_train, y_train)
```

```
In [88]: predicted = svc.predict(X_test)
```

```
In [89]: accuracy_score(y_test, predicted)
```

```
Out[89]: 0.8901515151515151
```

```
In [28]: from sklearn import tree
```

```
In [90]: decTree = tree.DecisionTreeClassifier().fit(X_train, y_train)
```

```
In [92]: predicted = decTree.predict(X_test)
```

```
In [93]: accuracy_score(y_test, predicted)
```

```
Out[93]: 0.634469696969697
```

```
In [102]: parameters = {'C':[0.5,1,2,3]}
```

```
In [103]: from sklearn.model_selection import GridSearchCV
```

```
In [104]: logist = LogisticRegression(random_state=42,multi_class='auto')
```

```
In [108]: logs = GridSearchCV(logist, parameters, cv=5, scoring='accuracy').fit(X_
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/model_selection/_sear
  DeprecationWarning)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/linear_model/logistic
  FutureWarning)
```

```
In [111]: logs.best_params_
```

```
Out[111]: {'C': 1}
```

```
In [109]: linsvc = LinearSVC(random_state=0, tol=1e-5)
In [112]: svcgv = GridSearchCV(linsvc, parameters, cv=5, scoring='accuracy').fit(X

In [113]: svcgv.best_params_

Out[113]: {'C': 0.5}

In [121]: svcc = LinearSVC(random_state=0, C=0.5, tol=1e-5).fit(X_train, y_train)

In [122]: predicted = svcc.predict(X_test)

In [123]: accuracy_score(y_test, predicted)

Out[123]: 0.8901515151515151

In [132]: parameters = {'max_depth':[10,20,30,50,100]}

In [127]: decTreeg = tree.DecisionTreeClassifier()

In [133]: dtgv = GridSearchCV(decTreeg, parameters, cv=5, scoring='accuracy').fit

/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/model_selection/_sea
  DeprecationWarning)


In [134]: dtgv.cv_results_

/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/utils/deprecation.py
  warnings.warn(*warn_args, **warn_kwargs)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/utils/deprecation.py
  warnings.warn(*warn_args, **warn_kwargs)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/utils/deprecation.py
  warnings.warn(*warn_args, **warn_kwargs)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/utils/deprecation.py
  warnings.warn(*warn_args, **warn_kwargs)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/utils/deprecation.py
  warnings.warn(*warn_args, **warn_kwargs)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/utils/deprecation.py
  warnings.warn(*warn_args, **warn_kwargs)
/home/hexagramg/exp/venv/lib/python3.6/site-packages/sklearn/utils/deprecation.py
  warnings.warn(*warn_args, **warn_kwargs)


Out[134]: {'mean_fit_time': array([0.58440495, 0.61546059, 0.61054664, 0.61003442
           'std_fit_time': array([0.01086698, 0.01959478, 0.01725378, 0.01792136,
           'mean_score_time': array([0.00217228, 0.00219789, 0.00223088, 0.0022218
           'std_score_time': array([4.16315706e-05, 2.51901838e-05, 3.51536047e-0
                  4.17281829e-05]),
           'param_max_depth': masked_array(data=[10, 20, 30, 50, 100],
                        mask=[False, False, False, False, False],
                  fill_value='?',
                       dtype=object),
           'params': [{'max_depth': 10},
            {'max_depth': 20},
            {'max_depth': 30},
            {'max_depth': 50},
            {'max_depth': 100}],
           'split0_test_score': array([0.5733945 , 0.62844037, 0.60550459, 0.60550
           'split1_test_score': array([0.56682028, 0.60368664, 0.56682028, 0.58064
           'split2_test_score': array([0.56542056, 0.57943925, 0.56074766, 0.58878
           'split3_test_score': array([0.60377358, 0.60849057, 0.58018868, 0.61792
           'split4_test_score': array([0.54285714, 0.54761905, 0.55714286, 0.52857
           'mean_test_score': array([0.57049486, 0.59383754, 0.57422969, 0.5845004
           'std_test_score': array([0.01945726, 0.02767944, 0.01762988, 0.03051168
           'rank_test_score': array([5, 1, 4, 2, 3], dtype=int32),
           'split0_train_score': array([0.92614302, 1.        , 1.        , 1.
           'split1_train_score': array([0.95433255, 1.        , 1.        , 1.
           'split2_train_score': array([0.97549592, 1.        , 1.        , 1.
           'split3_train_score': array([0.95343423, 1.        , 1.        , 1.
           'split4_train_score': array([0.91056911, 1.        , 1.        , 1.
           'mean_train_score': array([0.94399496, 1.        , 1.        , 1.
           'std_train_score': array([0.02290963, 0.        , 0.        , 0.
```