Letters

April 10, 2018

0.1 Letters Classification in Python

```
In [1]: #import libraries
        import time

        import numpy as np
        import pandas as pd
        from ggplot import *
        from matplotlib import pyplot as plt
        %matplotlib inline
        import seaborn as sns
        import random as random

        from sklearn.ensemble import RandomForestClassifier
        from sklearn.svm import SVC
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import GridSearchCV
```

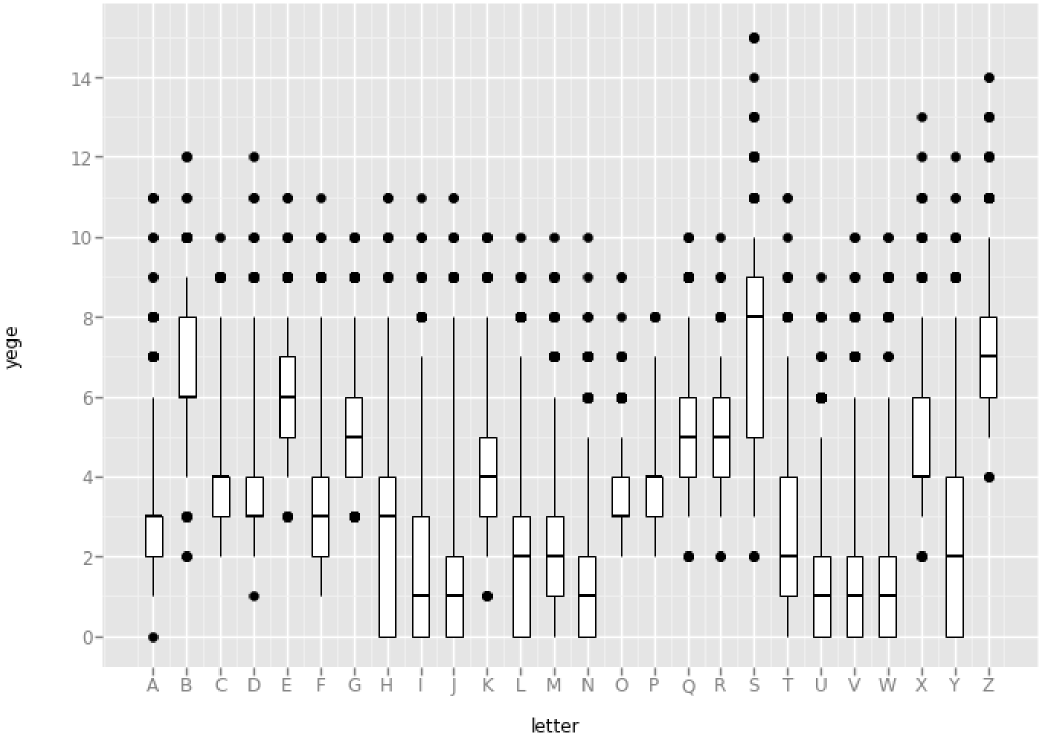
```
c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\ggplot\utils.py:81: FutureWarni
You can access Timestamp as pandas.Timestamp
  pd.tslib.Timestamp,
c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\ggplot\stats\smoothers.py:4: Fu
  from pandas.lib import Timestamp
c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\statsmodels\compat\pandas.py:56
  from pandas.core import datetools
```

```
In [2]: #read in the data
        data = pd.read_csv('https://mheaton.byu.edu/Courses/Stat536/Case%20Studies/LetterRecognition/Da
        data.head()
```

```
Out[2]:   letter  xbox  ybox  width  high  pix  xbar  ybar  x2bar  y2bar  xybar  \
        0      I     5    12      3     7    2    10     5      5      4     13
        1      D     4    11      6     8    6    10     6      2      6     10
        2      N     7    11      6     6    3     5     9      4      6      4
        3      G     2     1      3     1    1     8     6      6      6      6
        4      S     4    11      5     8    3     8     8      6      9      5

           x2ybar  xy2bar  xege  xegevy  yege  yegvx
        0       3       9     2       8     4     10
        1       3       7     3       7     3      9
        2       4      10     6      10     2      8
        3       5       9     1       7     5     10
        4       6       6     0       8     9      7
```

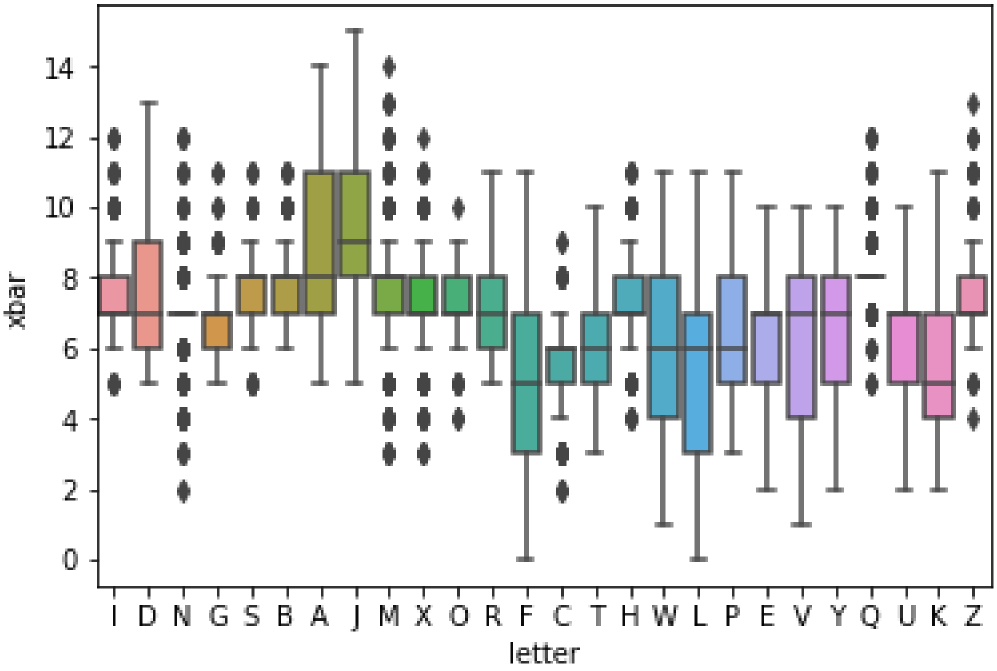
```
In [3]: #print(data.describe())
In [4]: #data.info()
```

```
In [5]: ggplot(aes(x='letter', y='yege'),data=data)+geom_boxplot()
```



```
Out[5]: <ggplot: (-9223371884800184106)>
```

```
In [6]: #quick summary statistics and plots
        # letters.boxplot('letter',by='xbox')
        # plt.show()

        sns.boxplot(y='xbar',x='letter',data=data)
        plt.show()
        # sns.boxplot(y='x2bar',x='letter',data=data)
        # plt.show()
        # sns.despine()
```



```
In [7]: # list(data.columns)
```

```
In [8]:  #sns.pairplot(data)
```

```
In [9]: #make test and train set for model
        X = data.drop('letter',axis=1)
        # X.head()
        y = data.letter

        X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25, random_state=123)
```

```
In [10]: #fit the model
         def my_model_train(model,X_tr,y_tr,X_tst,y_tst):
             train_start = time.time()
             model.fit(X_tr,y_tr)
             train_end = time.time()
             print('Training time:', train_end-train_start)
         #     benchmark = sum(y)/len(y) #should work if y is a vector of 0s and 1s
             score = model.score(X_tst, y_tst)
             print('Score:',score)
         #     print('Benchmark:',  benchmark,'\n')
```

```
In [11]: #different models
         mod_RF = RandomForestClassifier(n_estimators=1000, max_features=5)
         mod_SVM = SVC()
```

```
In [ ]:
```

```
In [12]: my_model_train(mod_RF,X_train,y_train,X_test,y_test)
```

```
Training time: 41.774861097335815
Score: 0.9696
```

```
In [13]: my_model_train(mod_SVM,X_train,y_train,X_test,y_test)
```

```
Training time: 15.264341592788696
Score: 0.9712
```

```
In [14]: ##Parameters to tune random forest
         numTrees=[10, 100, 1000, 2000]
         maxParms=[3,5,7,10]
         criterion=['gini', 'entropy']


         param_dict = dict(n_estimators=numTrees, max_features=maxParms, criterion=criterion)
         model=mod_RF

         #grid=GridSearchCV(cv=None, estimator=model, param_grid=param_dict)
         start = time.time()
         #grid.fit(X,y)
         end = time.time()
         runtime = end-start
         print('Minutes:',runtime/60)
```

```
        ---------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call last)

        <ipython-input-14-3361243490ec> in <module>()
         10 grid=GridSearchCV(cv=None, estimator=model, param_grid=param_dict)
         11 start = time.time()
    ---> 12 grid.fit(X,y)
         13 end = time.time()
         14 runtime = end-start


        c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\model_selection
        637                                     error_score=self.error_score)
        638           for parameters, (train, test) in product(candidate_params,
    --> 639                                                    cv.split(X, y, groups)))
        640
        641         # if one choose to see train score, "out" will contain train score info


        c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
        777             # was dispatched. In particular this covers the edge
        778             # case of Parallel used with an exhausted iterator.
    --> 779             while self.dispatch_one_batch(iterator):
        780                 self._iterating = True
        781             else:


        c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
        623                 return False
    624             else:
--> 625                 self._dispatch(tasks)
    626                 return True
    627 


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    586         dispatch_timestamp = time.time()
    587         cb = BatchCompletionCallBack(dispatch_timestamp, len(batch), self)
--> 588         job = self._backend.apply_async(batch, callback=cb)
    589         self._jobs.append(job)
    590 


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    109     def apply_async(self, func, callback=None):
    110         """Schedule a func to be run"""
--> 111         result = ImmediateResult(func)
    112         if callback:
    113             callback(result)


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    330         # Don’t delay the application, to avoid keeping the input
    331         # arguments in memory
--> 332         self.results = batch()
    333 
    334     def get(self):


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    129 
    130     def __call__(self):
--> 131         return [func(*args, **kwargs) for func, args, kwargs in self.items]
    132 
    133     def __len__(self):


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    129 
    130     def __call__(self):
--> 131         return [func(*args, **kwargs) for func, args, kwargs in self.items]
    132 
    133     def __len__(self):


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\model_selection
    490         if return_train_score:
    491             train_scores = _score(estimator, X_train, y_train, scorer,
--> 492                                   is_multimetric)
    493 
    494     if verbose > 2:
```

```
    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\model_selection
    521     """
    522     if is_multimetric:
--> 523         return _multimetric_score(estimator, X_test, y_test, scorer)
    524     else:
    525         if y_test is None:


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\model_selection
    551             score = scorer(estimator, X_test)
    552         else:
--> 553             score = scorer(estimator, X_test, y_test)
    554
    555         if hasattr(score, ’item’):


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\metrics\scorer.
    242 def _passthrough_scorer(estimator, *args, **kwargs):
    243     """Function that wraps estimator.score"""
--> 244     return estimator.score(*args, **kwargs)
    245
    246


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\base.py in scor
    347         """
    348         from .metrics import accuracy_score
--> 349         return accuracy_score(y, self.predict(X), sample_weight=sample_weight)
    350
    351


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\ensemble\forest
    536             The predicted classes.
    537         """
--> 538         proba = self.predict_proba(X)
    539
    540         if self.n_outputs_ == 1:


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\ensemble\forest
    587         Parallel(n_jobs=n_jobs, verbose=self.verbose, backend="threading")(
    588             delayed(accumulate_prediction)(e.predict_proba, X, all_proba, lock)
--> 589             for e in self.estimators_)
    590
    591         for proba in all_proba:


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    777             # was dispatched. In particular this covers the edge
    778             # case of Parallel used with an exhausted iterator.
--> 779             while self.dispatch_one_batch(iterator):
    780                 self._iterating = True
    781             else:
```

```
    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    623                 return False
    624             else:
--> 625                 self._dispatch(tasks)
    626                 return True
    627


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    586         dispatch_timestamp = time.time()
    587         cb = BatchCompletionCallBack(dispatch_timestamp, len(batch), self)
--> 588         job = self._backend.apply_async(batch, callback=cb)
    589         self._jobs.append(job)
    590


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    109     def apply_async(self, func, callback=None):
    110         """Schedule a func to be run"""
--> 111         result = ImmediateResult(func)
    112         if callback:
    113             callback(result)


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    330         # Don't delay the application, to avoid keeping the input
    331         # arguments in memory
--> 332         self.results = batch()
    333
    334     def get(self):


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    129
    130     def __call__(self):
--> 131         return [func(*args, **kwargs) for func, args, kwargs in self.items]
    132
    133     def __len__(self):


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    129
    130     def __call__(self):
--> 131         return [func(*args, **kwargs) for func, args, kwargs in self.items]
    132
    133     def __len__(self):


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\ensemble\forest
    384     with lock:
    385         if len(out) == 1:
--> 386             out[0] += prediction
```

```
387         else:
388             for i in range(len(out)):


KeyboardInterrupt:
```

Out of the box, it appears that SVM is better. Let's do a grid search of parameters to optimize this model.

```
In [15]: #parameters to tune
         k_list = ['rbf']#,'linear']#,'poly']#,'sigmoid']#,'precomputed']
         gam_list = ['auto']#,0.01,0.1,.5,1,10]
         c_list = [8,9,10,11,12,13]#,10,50]
         # deg_list = [1,2,3,4]#,5]

         #all together
         param_dict = dict(kernel=k_list,gamma=gam_list,C = c_list)#,degree=deg_list)


         #model
         model = mod_SVM
         # model = mod_RF

         grid = GridSearchCV(cv=None,estimator=model, param_grid=param_dict)
         start = time.time()
         grid.fit(X,y)
         end = time.time()
         runtime = end-start
         print('Minutes:',runtime/60)
```

```
        ---------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call last)

        <ipython-input-15-c6fe82faf6e5> in <module>()
         15 grid = GridSearchCV(cv=None,estimator=model, param_grid=param_dict)
         16 start = time.time()
    ---> 17 grid.fit(X,y)
         18 end = time.time()
         19 runtime = end-start


        c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\model_selection
        637                                     error_score=self.error_score)
        638           for parameters, (train, test) in product(candidate_params,
    --> 639                                                     cv.split(X, y, groups)))
        640
        641         # if one choose to see train score, "out" will contain train score info


        c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
        777             # was dispatched. In particular this covers the edge
        778             # case of Parallel used with an exhausted iterator.
```

```
--> 779             while self.dispatch_one_batch(iterator):
    780                 self._iterating = True
    781             else:


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    623                 return False
    624             else:
--> 625                 self._dispatch(tasks)
    626                 return True
    627


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    586         dispatch_timestamp = time.time()
    587         cb = BatchCompletionCallBack(dispatch_timestamp, len(batch), self)
--> 588         job = self._backend.apply_async(batch, callback=cb)
    589         self._jobs.append(job)
    590


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    109     def apply_async(self, func, callback=None):
    110         """Schedule a func to be run"""
--> 111         result = ImmediateResult(func)
    112         if callback:
    113             callback(result)


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    330         # Don't delay the application, to avoid keeping the input
    331         # arguments in memory
--> 332         self.results = batch()
    333
    334     def get(self):


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    129
    130     def __call__(self):
--> 131         return [func(*args, **kwargs) for func, args, kwargs in self.items]
    132
    133     def __len__(self):


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\externals\jobli
    129
    130     def __call__(self):
--> 131         return [func(*args, **kwargs) for func, args, kwargs in self.items]
    132
    133     def __len__(self):


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\model_selection
    490         if return_train_score:
    491             train_scores = _score(estimator, X_train, y_train, scorer,
--> 492                                   is_multimetric)
    493
    494     if verbose > 2:


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\model_selection
    521     """
    522     if is_multimetric:
--> 523         return _multimetric_score(estimator, X_test, y_test, scorer)
    524     else:
    525         if y_test is None:


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\model_selection
    551             score = scorer(estimator, X_test)
    552         else:
--> 553             score = scorer(estimator, X_test, y_test)
    554
    555         if hasattr(score, 'item'):


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\metrics\scorer.
    242 def _passthrough_scorer(estimator, *args, **kwargs):
    243     """Function that wraps estimator.score"""
--> 244     return estimator.score(*args, **kwargs)
    245
    246


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\base.py in scor
    347         """
    348         from .metrics import accuracy_score
--> 349         return accuracy_score(y, self.predict(X), sample_weight=sample_weight)
    350
    351


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\svm\base.py in
    546             Class labels for samples in X.
    547         """
--> 548         y = super(BaseSVC, self).predict(X)
    549         return self.classes_.take(np.asarray(y, dtype=np.intp))
    550


    c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\svm\base.py in
    308         X = self._validate_for_predict(X)
    309         predict = self._sparse_predict if self._sparse else self._dense_predict
--> 310         return predict(X)
    311
    312     def _dense_predict(self, X):
```

```
c:\users\jntrcs\appdata\local\programs\python\python35\lib\site-packages\sklearn\svm\base.py in
331                self.probA_, self.probB_, svm_type=svm_type, kernel=kernel,
332                degree=self.degree, coef0=self.coef0, gamma=self._gamma,
--> 333                cache_size=self.cache_size)
334
335        def _sparse_predict(self, X):


KeyboardInterrupt:
```

```
In [ ]: print("Best Score:",grid.best_score_)
        # print("Kernel:",grid.best_estimator_.kernel)
        # print("Gamma:",grid.best_estimator_.gamma)
        # print("C:",grid.best_estimator_.C)
        print('Best Params:',grid.best_params_)
```

```
In [16]: temp_mod = SVC(gamma='auto',C=11,kernel='rbf')
         my_model_train(temp_mod,X_train,y_train,X_test,y_test)
```

```
Training time: 15.484598875045776
Score: 0.9786
```

```
In [ ]: #notes of best models
        # Best Score: 0.9742987149357468
        # Best Params: {'gamma': 'auto', 'kernel': 'rbf', 'C': 5}
```

```
In [17]: # make a confusion matrix and plot it (if possible)
         from sklearn.metrics import confusion_matrix
         import itertools
```

```
In [19]: data['rand']=np.random.choice([1, 2, 3, 4, 5],  19999)
         data.loc[:,'pred']="A"
         data.head
```

```
Out[19]: <bound method NDFrame.head of       letter  xbox  ybox  width  high  pix  xbar  ybar  x2bar  y
         0          I     5    12      3     7    2    10     5      5     4    13
         1          D     4    11      6     8    6    10     6      2     6    10
         2          N     7    11      6     6    3     5     9      4     6     4
         3          G     2     1      3     1    1     8     6      6     6     6
         4          S     4    11      5     8    3     8     8      6     9     5
         5          B     4     2      5     4    4     8     7      6     6     7
         6          A     1     1      3     2    1     8     2      2     2     8
         7          J     2     2      4     4    2    10     6      2     6    12
         8          M    11    15     13     9    7    13     2      6     2    12
         9          X     3     9      5     7    4     8     7      3     8     5
         10         O     6    13      4     7    4     6     7      6     3    10
         11         G     4     9      6     7    6     7     8      6     2     6
         12         M     6     9      8     6    9     7     8      6     5     7
         13         R     5     9      5     7    6     6    11      7     3     7
         14         F     6     9      5     4    3    10     6      3     5    10
         15         O     3     4      4     3    2     8     7      7     5     7
         16         C     7    10      5     5    2     6     8      6     8    11
         17         T     6    11      6     8    5     6    11      5     6    11
         18         J     2     2      3     3    1    10     6      3     6    12
```

```
19         J   1   3    2   2   1    8    8    2    5   14
20         H   4   5    5   4   4    7    7    6    6    7
21         S   3   2    3   3   2    8    8    7    5    7
22         O   6  11    7   8   5    7    6    9    6    7
23         J   3   6    4   4   2    6    6    4    4   14
24         C   6  11    7   8   3    7    8    7   11    4
25         M   7  11   11   8   9    3    8    4    5   10
26         W  12  14   12   8   5    9   10    4    3    5
27         H   6   9    8   7   6    8    6    6    7    7
28         G   3   6    4   4   2    6    6    5    5    6
29         L   2   3    3   4   1    0    1    5    6    0
...      ... ...  ...  ... ... ...  ...  ...  ...  ...  ...
19969      F   7  10    9   8   7    9    7    2    6   12
19970      C   5  10    7   9   8    5    6    4    4    7
19971      V   4   7    6   5   6    8    6    4    2    7
19972      T   4   4    5   3   2    5   12    2    8   11
19973      N   5   9    5   4   2    9   11    5    3    5
19974      E   1   0    1   0   0    5    8    5    7    7
19975      L   3   8    3   6   2    0    2    4    6    1
19976      A   3   9    5   6   2    6    5    3    1    6
19977      K   5  11    5   8   5    3    8    7    3    6
19978      M   6   9   10   7  12    7    5    3    2    7
19979      R   2   3    3   2   2    7    7    5    5    7
19980      S   6  12    6   7   3    6    8    3    6   13
19981      Y   3   9    5   6   3    7    9    1    6    6
19982      V   7  10    5   5   2    6   11    5    4   11
19983      S   2   0    2   1   1    8    7    4    6    5
19984      M   5   6    8   4   5    9    6    2    4    9
19985      O   9  15    6   8   5    5    7    7    4   10
19986      L   3   7    3   5   1    0    1    6    6    0
19987      D   6   9    8   8   8    7    6    5    7    7
19988      P   2   1    3   2   1    4   10    3    5   10
19989      W   3   8    5   6   5   11   11    2    2    5
19990      O   4   3    5   4   2    7    6    8    8    6
19991      E   4   9    5   6   3    5    9    2   10   10
19992      J   2  11    3   8   2   15    4    4    5   13
19993      T   5   8    7   7   7    7    9    4    8    7
19994      D   2   2    3   3   2    7    7    7    6    6
19995      C   7  10    8   8   4    4    8    6    9   12
19996      T   6   9    6   7   5    6   11    3    7   11
19997      S   2   3    4   2   1    8    7    2    6   10
19998      A   4   9    6   6   2    9    5    3    1    8

       x2ybar  xy2bar  xege  xegevy  yege  yegvx  rand pred
0           3       9     2       8     4     10     3    A
1           3       7     3       7     3      9     2    A
2           4      10     6      10     2      8     4    A
3           5       9     1       7     5     10     1    A
4           6       6     0       8     9      7     2    A
5           6       6     2       8     7     10     2    A
6           2       8     1       6     2      7     2    A
7           4       8     1       6     1      7     5    A
8           1       9     8       1     1      8     1    A
9           6       8     2       8     6      7     1    A
```

```
10          7       9     5       9     5      8     3    A
11          5      11     4       8     7      8     3    A
12          5       8     8       9     8      6     4    A
13          3       9     2       7     5     11     3    A
14          5       7     3       9     6      9     2    A
15          6       8     2       8     3      8     5    A
16          7      11     2       8     5      9     1    A
17          9       4     3      12     2      4     4    A
18          4       9     0       7     1      7     4    A
19          5       8     0       7     0      7     1    A
20          6       8     3       8     3      8     2    A
21          5       7     2       8     9      8     3    A
22          5       9     4       8     5      5     3    A
23          8      12     1       6     1      6     2    A
24          7      14     1       7     4      8     4    A
25         11      10    10       9     5      7     2    A
26         10       7    10      12     2      6     1    A
27          7       9     6       8     4      8     3    A
28          6       9     2       8     4      8     4    A
29          0       6     0       8     0      8     4    A
...       ...     ...   ...     ...   ...    ...   ...  ...
19969       4       6     5       9     4      9     5    A
19970       6      11     5      11     8     10     5    A
19971       8       8     7       9     4      6     1    A
19972       9       4     0      10     2      4     4    A
19973       6       9     5      11     2      6     5    A
19974       6      12     0       8     6     10     5    A
19975       0       8     0       8     0      8     3    A
19976       1       8     2       7     2      7     3    A
19977       4      11     3       8     2     11     5    A
19978       5       8    15       7     4      6     5    A
19979       5       6     2       7     4      8     2    A
19980       7       7     2       9     3      7     2    A
19981      11       8     2      11     2      7     2    A
19982       9       4     4      11     3     10     4    A
19983       6       8     0       8     7      8     4    A
19984       5       7     8       6     2      8     3    A
19985       7      10     5       9     5      8     3    A
19986       0       6     0       8     0      8     5    A
19987       5       9     6       5    10      3     3    A
19988       8       5     0       9     3      7     5    A
19989       8       7     7      12     1      7     3    A
19990       5       7     3       8     4      8     1    A
19991       8       9     2       8     5      5     3    A
19992       1       8     0       7     0      8     3    A
19993       7       8     3      10     8      6     4    A
19994       6       4     2       8     3      7     3    A
19995       9      13     2       9     3      7     2    A
19996       9       5     2      12     2      4     4    A
19997       6       8     1       9     5      8     2    A
19998       1       8     2       7     2      8     2    A

[19999 rows x 19 columns]>
```

```
In [20]: #best model

         for i in range(1,6):
             print(i)
             X_test=data.loc[data.rand==i, 'xbox':'yegvx']   #X_tr.head
             X_train=data.loc[data.rand!=i, 'xbox':'yegvx']
             y_train=data.loc[data.rand!=i, 'letter']
             y_test = data.loc[data.rand==i, 'letter']
             model_best = SVC(gamma = 'auto', kernel = 'rbf', C = 5)
             my_model_train(model_best,X_tr=X_train,y_tr=y_train,X_tst=X_test,y_tst=y_test)
             y_pred = model_best.predict(X_test)
             data.loc[data.rand==i, 'pred']=y_pred
         #compute matrix
         #y_pred = model_best.predict(X_test)
         #cnf_matrix = confusion_matrix(y_test,y_pred)

         # print(y_pred)
         # print(y_test)
         # print(np.sum(y_pred == y_test))
         # print(cnf_matrix)
```

```
1
Training time: 17.65030860900879
Score: 0.9761784085149519
2
Training time: 16.399207592010498
Score: 0.9736114601658709
3
Training time: 17.20598006248474
Score: 0.9768714250186521
4
Training time: 17.690088272094727
Score: 0.9812879708383961
5
Training time: 16.911091566085815
Score: 0.9771457592686643
```

```
In [29]: data=data.sort_values(by="letter")
         cnf_matrix = confusion_matrix(data.loc[:,'letter'],data.loc[:,'pred'])
```

```
<bound method NDFrame.head of       letter  xbox  ybox  width  high  pix  xbar  ybar  x2bar  y2bar  xyba
19998      A     4     9      6     6    2     9     5      3      1      8
17854      A     6    10      6     5    4    12     3      6      2     12
11447      A     3     7      5     6    4     7     8      2      4      7
1076       A     3     7      5     5    3    11     2      2      2      9
11530      A     3     9      6     7    4    11     3      1      2      8
4300       A     3     7      5     5    3    11     2      3      2     10
11573      A     3     7      5     5    2    12     3      4      3     11
11605      A     2     2      4     4    2     8     2      2      2      8
17807      A     3     8      4     6    3     8     3      2      2      7
11620      A     3    11      5     8    3    13     4      5      3     12
4256       A     5     6      7     5    6     6     6      3      5      7
11645      A     5    10      9     7    6     7     5      2      4      5
11647      A     4     9      6     6    3    11     2      3      3     10
4244       A     3     2      6     4    2    10     2      2      2      9
```

```
4241       A     5     8      7     6    6      8     9      7      5      6
17783      A     3     8      5     5    2      9     6      3      1      7
11654      A     3     8      5     5    2      7     4      3      1      7
4224       A     1     0      2     0    0      8     4      2      0      7
11849      A     4     9      5     6    5      7     6      7      4      7
17697      A     5     9      5     5    3     10     2      4      2     11
17699      A     4    11      7     8    2      7     5      3      1      6
11807      A     4     7      6     6    5      8     8      2      4      7
11798      A     5    10      7     7    8      8     5      8      4      8
11772      A     3     8      5     6    3     10     4      2      2      8
11406      A     6    10      8     8    8      7     8      8      4      6
11763      A     5     7      7     5    7      7     8      8      4      7
4185       A     4     8      6     6    3     13     3      4      3     11
1138       A     3     3      5     4    1      8     6      3      1      7
11721      A     3     5      5     4    4      7     8      3      4      7
11707      A     7    10      9     8    9      8     6      7      4      7
...       ...   ...   ...    ...   ...  ...    ...   ...    ...    ...    ...
11478      Z     4     7      6     5    3      6     9      3      9     12
4323       Z     5    10      6     8    5      7     8      3     12      9
17463      Z     3     8      4     6    3      9     6      5     10      7
11465      Z     2     7      3     5    2      6     8      5     10      6
2986       Z     4    10      5     8    5      8     7      6     10      7
14123      Z     3     5      5     4    3      8     7      2      9     12
1053       Z     3     5      4     7    2      7     7      4     14     10
11370      Z     6     7      8     9    8      9     8      6      5      9
14112      Z     6     9      8     7    6      6     9      2      9     11
11380      Z     1     0      1     0    0      7     7      2      9      8
9030       Z     1     0      1     0    0      8     7      2      9      8
9026       Z     4    10      6     7    4      9     5      3     10     11
11385      Z     6    11      8     9    8     10     7      5      4      7
9020       Z     5    11      7     8    4      7     7      2     10     12
18672      Z     4     9      5     7    5      8     8      3      8      7
15600      Z     4    10      5     8    3      7     7      4     15      9
9007       Z     5     8      7    10    6     11     4      3      5      9
9006       Z     5     5      6     8    3      7     7      4     15      9
9005       Z     5    11      7     9    4      8     7      2     10     11
17884      Z     5     9      6     4    3     10     3      3      7     12
19351      Z     3     7      4     5    4      6     6      3      7      7
15622      Z     2     4      4     3    2      7     8      2      9     11
17889      Z     7    11      7     6    4      8     6      2      8     11
17467      Z     3     2      4     4    2      7     7      5     10      6
15639      Z     5     8      7     6    4      6     9      3     10     11
4326       Z     4     8      5     6    5      8     8      3      7      7
15669      Z     4    11      6     8    7      8     7      2      8      7
4324       Z     2     5      4     4    2      7     8      2     10     11
11394      Z     3     2      4     3    2      7     7      5      9      6
3405       Z     7    11      9     8    7      7     7      2      9     12

       x2ybar  xy2bar  xege  xegevy  yege  yegvx  rand pred
19998       1       8     2       7     2      8     2    A
17854       2      10     5       3     3     10     1    A
11447       8       9     5       8     3      6     4    A
1076        2       9     3       6     3      9     4    A
11530       3       9     4       5     3      8     4    A
```

4300	2	9	3	7	3	9	5	A
11573	2	10	2	6	3	9	3	A
11605	2	8	2	6	3	7	4	A
17807	1	8	2	6	2	7	1	A
11620	1	8	2	6	4	9	2	A
4256	8	10	8	11	3	8	5	A
11645	1	6	5	7	5	5	4	A
11647	2	9	2	6	3	8	3	A
4244	1	8	2	6	2	8	4	A
4241	6	8	3	7	7	4	5	A
17783	0	8	2	7	1	8	4	A
11654	1	8	3	7	2	8	1	A
4224	2	8	1	6	1	8	2	A
11849	6	9	2	8	8	4	1	A
17697	5	12	5	3	5	10	2	A
17699	1	8	3	7	2	7	5	A
11807	7	8	5	7	4	6	4	A
11798	6	8	3	9	8	3	3	A
11772	2	10	2	6	3	8	4	A
11406	5	9	3	7	8	5	1	A
11763	5	8	4	8	9	4	5	A
4185	1	8	2	6	2	9	4	A
1138	1	8	2	7	1	8	4	A
11721	8	8	5	10	3	6	5	A
11707	6	9	6	8	8	3	3	A
...	...	...	...	...	...	...	...	...
11478	9	7	1	9	6	5	5	Z
4323	6	8	0	8	8	7	2	Z
17463	5	6	1	7	8	8	4	Z
11465	7	9	1	9	8	8	1	Z
2986	5	7	1	7	8	8	3	Z
14123	6	9	1	8	5	7	4	Z
1053	6	8	0	8	8	8	5	Z
11370	3	6	3	5	8	7	1	Z
14112	8	7	3	11	7	7	5	Z
11380	6	8	0	8	6	8	4	Z
9030	6	8	0	8	5	8	2	Z
9026	4	9	1	7	6	9	5	Z
11385	5	7	4	8	10	5	3	Z
9020	5	9	1	9	7	8	2	Z
18672	7	7	1	8	11	7	4	Z
15600	6	8	0	8	8	8	3	Z
9007	2	7	2	7	6	9	3	Z
9006	6	8	0	8	8	8	2	Z
9005	5	9	2	8	6	8	1	Z
17884	4	10	2	9	4	11	1	Z
19351	6	10	1	7	10	7	3	Z
15622	6	8	1	8	5	7	4	Z
17889	6	9	3	9	5	8	3	Z
17467	6	8	1	8	7	8	3	Z
15639	9	5	1	8	6	5	3	Z
4326	6	7	1	9	10	8	1	Z
15669	6	7	1	7	11	7	3	Z
4324	6	7	1	8	6	7	1	Z

```
11394       6       8     1       8     7      8     2    Z
3405        6       9     2       9     6      8     3    Z

[19999 rows x 19 columns]>
```

```
In [39]: print(np.mean(data.letter==data.pred)) #overall accuracy
         letters=list(string.ascii_uppercase)
         pd.DataFrame({"Letter":letters, "Accuracy":np.diag(cnf_matrix)/np.sum(cnf_matrix, axis=1)}).so
```

```
0.9770488524426222
```

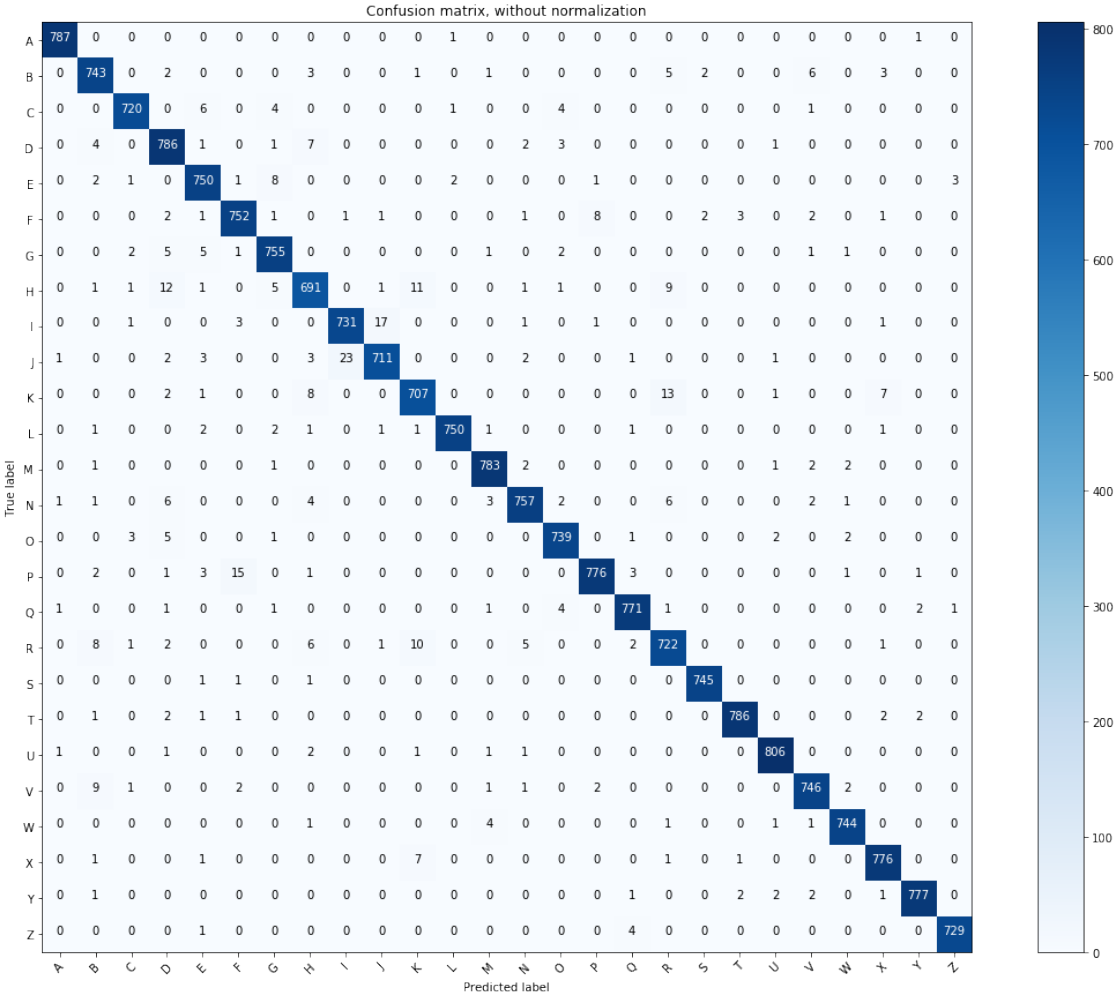
```
Out[39]:     Accuracy Letter
         0   0.997465      A
         18  0.995989      S
         25  0.993188      Z
         20  0.991390      U
         22  0.989362      W
         19  0.988679      T
         12  0.988636      M
         24  0.988550      Y
         23  0.986023      X
         11  0.985545      L
         16  0.984674      Q
         14  0.981408      O
         2   0.978261      C
         6   0.976714      G
         4   0.976562      E
         21  0.976440      V
         3   0.976398      D
         5   0.970323      F
         1   0.969974      B
         8   0.968212      I
         13  0.966794      N
         15  0.966376      P
         10  0.956698      K
         17  0.952507      R
         9   0.951807      J
         7   0.941417      H
```

```
In [23]: def plot_confusion_matrix(cm, classes,
                                   normalize=False,
                                   title='Confusion matrix',
                                   cmap=plt.cm.Blues):
             """
             This function prints and plots the confusion matrix.
             Normalization can be applied by setting `normalize=True`.
             """
             if normalize:
                 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
                 print("Normalized confusion matrix")
             else:
                 print('Confusion matrix, without normalization')

         #     print(cm)
```

```
            plt.imshow(cm, interpolation='nearest', cmap=cmap)
            plt.title(title)
            plt.colorbar()
            tick_marks = np.arange(len(classes))
            plt.xticks(tick_marks, classes, rotation=45)
            plt.yticks(tick_marks, classes)

            fmt = '.2f' if normalize else 'd'
            thresh = cm.max() / 2.
            for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                plt.text(j, i, format(cm[i, j], fmt),
                         horizontalalignment="center",
                         color="white" if cm[i, j] > thresh else "black")

            plt.tight_layout()
            plt.ylabel('True label')
            plt.xlabel('Predicted label')
```

```
In [32]: #plot non_normalized confusion Matrix
         # class_names = ['Bad','Good']
         import string
         class_names = list(string.ascii_uppercase)

         plt.figure(figsize=(16, 12))
         plot_confusion_matrix(cnf_matrix, classes=class_names,
                               title='Confusion matrix, without normalization')
         plt.show()
```

```
Confusion matrix, without normalization
```



```
In [ ]: # list(range(1,27))
```

```
In [ ]: # list(string.ascii_lowercase)
```

0.1.1 Make an example visualization of an SVM

- split off the letters a,b,c, and d and classify those with xbar and ybar

```
In [ ]: #make modified dataset
        # mini_dat = data
        # print(data.head(5))
        # data.iloc[0,0:3]
        mini_data = data[data['letter'].isin(['A','B','C'])][['letter','xbar','ybar']]#,['letter','xbar
        # mini_data = mini_data[['letter','xbar','ybar']]
        mini_data.head()
```

```
In [ ]: mini_data.info()
        mini_data.letter.describe()
```

```
In [ ]:
```

```
In [ ]: #plot the decision region
        from matplotlib.colors import ListedColormap

        def plot_decision_regions(X,y,classifier,test_idx = None, resolution = 0.02):
            #setup marker generator and color map
            markers = ['s','x','o','v','^']
            colors = ('red','blue','lightgreen','gray','cyan')
            cmap = ListedColormap(colors[:len(np.unique(y))])

            #plot the decision surface
            x1_min,x1_max = X[:,0].min() - 1, X[:,0].max() + 1 #sepal length
            x2_min,x2_max = X[:,1].min() - 1, X[:,1].max() + 1 #petal length

            xx1, xx2 = np.meshgrid(np.arange(x1_min,x1_max,resolution),np.arange(x2_min,x2_max,resoluti

            Z = classifier.predict(np.array([xx1.ravel(),xx2.ravel()]).T)
            Z = Z.reshape(xx1.shape)

            plt.contourf(xx1,xx2,Z,alpha = 0.4,cmap = cmap)
            plt.xlim(xx1.min(), xx1.max())
            plt.ylim(xx2.min(), xx2.max())

            #plot all samples
            for idx, cl in enumerate(np.unique(y)):
                plt.scatter(x = X[y==cl,0],y = X[y==cl,1],alpha = 0.8,c=cmap(idx),marker = markers[idx]
```

```
In [ ]: #set up a mini svm for visualization
        predictors = mini_data.drop('letter',axis=1)
        temp = mini_data.letter
        empt = []
        # print(temp)
        for i in temp:
        #
            if i == 'A':
                i = 0
            elif i == 'B':
                i = 1
            else:
                i = 2
            empt.append(i)
        target = empt
        target[:20]
```

```
In [ ]: #mini model
        p1,p2,t1,t2 = train_test_split(predictors,target,test_size=0.25, random_state=123)
        my_model_train(mod_SVM,p1,t1,p2,t2)
        mini_mod = SVC().fit(predictors,target)
```

```
In [ ]: # plot_decision_regions(X = np.array(predictors), y = np.array(target), classifier=model_best)
        plot_decision_regions(X = np.array(predictors), y = np.array(target), classifier=mini_mod)
        plt.xlabel('xbar')
        plt.ylabel('ybar')
        plt.legend( loc = 'upper right')
        L=plt.legend()
        L.get_texts()[0].set_text('A')
        L.get_texts()[1].set_text('B')
        L.get_texts()[2].set_text('C')
        plt.savefig('DemoSVM.pdf')

        plt.show()
```

```
In [ ]:

In [ ]:

In [ ]:
```