

# Contents

1	<a href="#">Ex3 Answers.pdf</a>	2
2	<a href="#">./comparsion.py</a>	6
3	<a href="#">./mnsit data.py</a>	9
4	<a href="#">./models.py</a>	11

# למידת מכונה 5

14 במאי 2020

## 1 תרגיל 1 :

הגדרנו את  $h_D(x) = \begin{cases} 1 & \mathcal{P}(y=1|x) \geq \frac{1}{2} \\ -1 & \text{otherwise} \end{cases}$ . נזכר כי לפי כלל בייס  $\mathcal{P}(x|y)\mathcal{P}(y) = \mathcal{P}(y|x)\mathcal{P}(x)$  מצד שני מאחר ו  $\sum \mathcal{P}(y|x) = 1$  ו-  
 $\mathcal{P}(y=1|x) \geq \frac{1}{2}$  אז מתקיים כי אם  $y \in \{-1, 1\}$  אז  $\mathcal{P}(y=1|x) = \max_{y \in \{-1, 1\}} \{\mathcal{P}(y|x)\}$  ו- $\mathcal{P}(y=1|x) \geq \frac{1}{2}$  באותו אופן עבור עברור  $\mathcal{P}(y=-1|x) \geq \frac{1}{2}$ .  
 ולכן:

$$\begin{aligned} h_D(x) &= \arg \max_{y \in \{-1, 1\}} \mathcal{P}(y|x) = \mathcal{P}(x) \arg \max_{y \in \{-1, 1\}} \mathcal{P}(y|x) \stackrel{(1)}{=} \\ &= \arg \max_{y \in \{-1, 1\}} \mathcal{P}(y|x)\mathcal{P}(x) = \arg \max_{y \in \{-1, 1\}} \mathcal{P}(x|y)\mathcal{P}(y) \end{aligned}$$

כאשר הכנסנו את  $x$  מאחר והוא מגדיל את כל האיברים מהם אנו מחלצים את המקסימום באופן זהה.

## 2 תרגיל 2 :

ממונטוניות חזקה של פונקציית ה  $\ln$  נקבל כי  $\arg \max_y g(y) = \arg \max_y \ln(g(y))$  ולכן

$$\begin{aligned} \Rightarrow h_D(x) &= \arg \max_{y \in \{-1, 1\}} \mathcal{P}(x|y)\mathcal{P}(y) = \arg \max_{y \in \{-1, 1\}} \ln(\mathcal{P}(x|y)\mathcal{P}(y)) = \\ &= \arg \max_{y \in \{-1, 1\}} \left\{ \ln \left( \frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} e^{-\frac{1}{2}(x-\mu_y)^T \Sigma^{-1}(x-\mu_y)} \right) + \ln(\mathcal{P}(y)) \right\} = \\ &= \arg \max_{y \in \{-1, 1\}} \left\{ \underbrace{\ln \left( \frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} \right)}_{\text{not depend at } y} + \ln \left( e^{-\frac{1}{2}(x-\mu_y)^T \Sigma^{-1}(x-\mu_y)} \right) + \ln(\mathcal{P}(y)) \right\} = \\ &= \arg \max_{y \in \{-1, 1\}} \left\{ \ln \left( e^{-\frac{1}{2}(x-\mu_y)^T \Sigma^{-1}(x-\mu_y)} \right) + \ln(\mathcal{P}(y)) \right\} = \arg \max_{y \in \{-1, 1\}} \left\{ -\frac{1}{2}(x-\mu_y)^T \Sigma^{-1}(x-\mu_y) + \ln(\mathcal{P}(y)) \right\} = \\ &= \arg \max_{y \in \{-1, 1\}} \left\{ \underbrace{-\frac{1}{2}x^T \Sigma^{-1}x + x^T \Sigma^{-1}\mu_y - \frac{1}{2}\mu_y^T \Sigma^{-1}\mu_y}_{\text{not depend at } y} + \ln(\mathcal{P}(y)) \right\} = \\ &= \arg \max_{y \in \{-1, 1\}} \left\{ x^T \Sigma^{-1}\mu_y - \frac{1}{2}\mu_y^T \Sigma^{-1}\mu_y + \ln(\mathcal{P}(y)) \right\} = \arg \max_{y \in \{-1, 1\}} \delta_y \end{aligned}$$

## 3 תרגיל 3 :

את  $\Sigma$  ו  $\mathcal{P}$  נחשב על ידי החישוב הרגיל על  $training - samples$ . (חישוב ישיר) מה שמשתנה ולא היה בשאלות עד כה זה החישוב של  $\mu_+$  ו  $\mu_-$  אבל גם  
 כאן זה מאוד פשוט.  $\mu_{\pm} = \text{Mean}\{x | (x, y) = (x, \pm 1)\}$ .

## 4 תרגיל 4 :

נירצה שהבחינה של הודעה נורמטיבת כהודעת ספאם תיחיה :  $Type - 2 - error$ . ולכן נתייג כ  $\{not - spam, true\} = False Positive$ . ואת :  
 $\{spam, false\} = True Positive$ .

## 5 תרגיל 5

אני מצרף תמונה בכתב יד מאחר וזה הרבה יותר ברור ככה :

SVM - Formulation 5

(יחסית ל-  $Q=2I$  ,  $a=0$  ,  $b=1$  )

$$\forall_j \quad y_j \cdot (\langle \omega, x_j \rangle + b) \geq 1$$

$$\Leftrightarrow y_j \langle \omega, x_j \rangle \geq 1 - b y_j$$

$$\Rightarrow \begin{bmatrix} -y_1 \\ -y_2 \\ \vdots \\ -y_m \end{bmatrix} \begin{bmatrix} -\omega \\ -\omega \\ \vdots \\ -\omega \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix} \geq \begin{bmatrix} 1 - b y_1 \\ \vdots \\ 1 - b y_m \end{bmatrix}$$

$$\Rightarrow \underbrace{\begin{bmatrix} -y_1 & -y_2 & \dots & -y_m \end{bmatrix}}_A \underbrace{\begin{bmatrix} -\omega \\ -\omega \\ \vdots \\ -\omega \end{bmatrix}}_v \underbrace{\begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix}}_d \leq \underbrace{\begin{bmatrix} 1 - b y_1 \\ \vdots \\ 1 - b y_m \end{bmatrix}}_d$$

$A = -y^T$   $\uparrow$   $\omega$   
 (מכאן  $\omega$  הוא וקטור המישור)

## 6 תרגיל 6

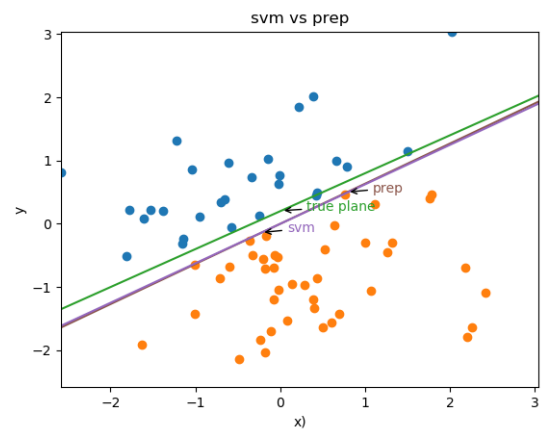
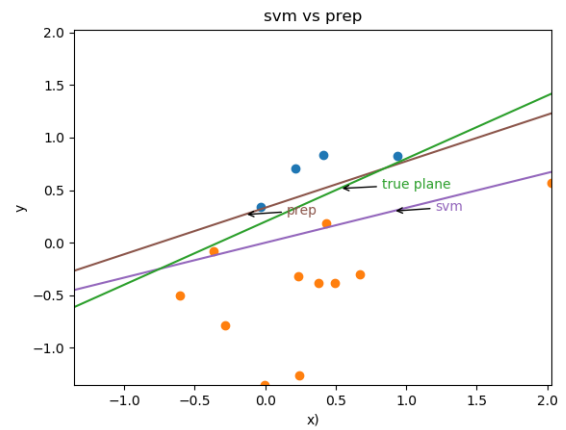
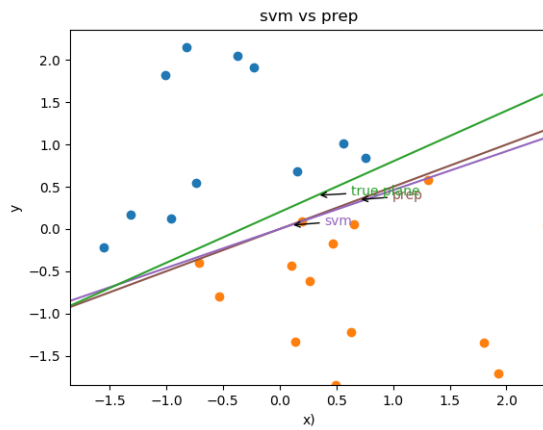
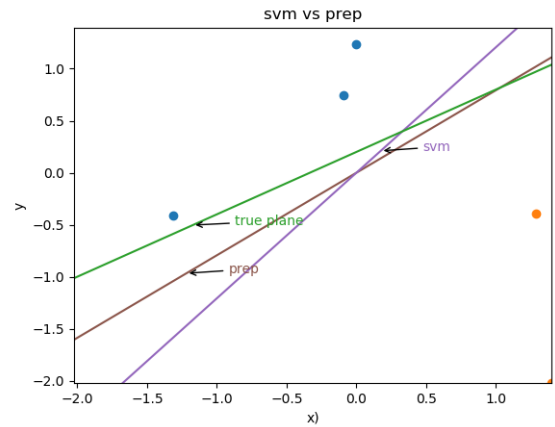
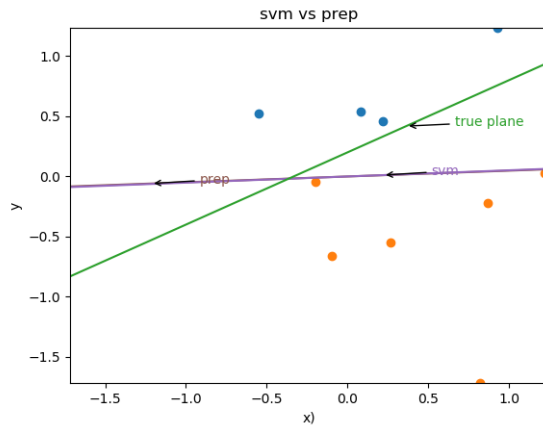
נראה שקילות בין שתי הבעיות :

$$\begin{aligned} \min_{\omega, \xi_i, y_i \langle \omega, x_i \rangle \geq 1 - \xi_i} \frac{\lambda}{2} \|\omega\|^2 + \frac{1}{m} \sum_{i=1}^m \xi_i &= \min_{\omega, \omega, \xi_i, y_i \langle \omega, x_i \rangle \geq 1 - \xi_i} \left\{ \frac{\lambda}{2} \|\omega\|^2 + \frac{1}{m} \sum_{i=1}^m \min_{\xi_i} \xi_i \right\} = \\ &= \min \left\{ \frac{\lambda}{2} \|\omega\|^2 + \frac{1}{m} \sum_{i=1}^m \begin{cases} 1 - y_i \langle \omega, x_i \rangle & \text{if } 1 - y_i \langle \omega, x_i \rangle \geq 0 \\ 0 & \text{otherwise} \end{cases} \right\} = \\ &= \min_{\omega} \left\{ \frac{\lambda}{2} \|\omega\|^2 + \frac{1}{m} \sum_{i=1}^m \max \{0, 1 - y_i \langle \omega, x_i \rangle\} \right\} \end{aligned}$$

נשים לב שכל מעבר הוא בשיוון ולכן הבכרח פתרון של בעיה אחת פותר גם את הבעיה השניה.

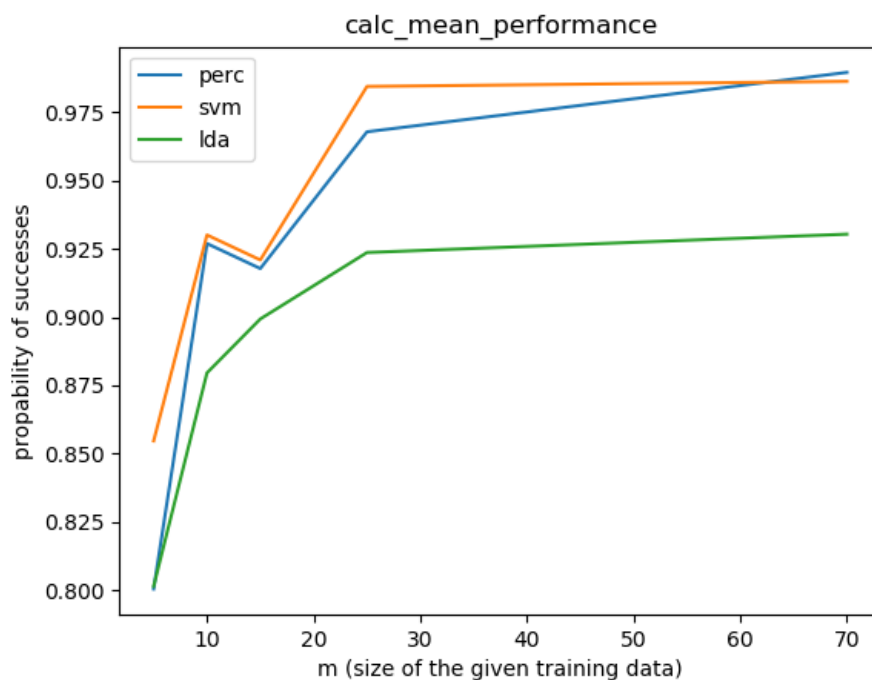
## 7 השוואה בין שיטות למציאת ישר מפריד.

כל השיטות נמצאו יעילות מבחינת יכולת הדיוק, נציין ש  $LDA$  איטית באופן משמעותי, בעיקר השלב של מציאת המטריצה ההופכית ( $\Sigma^\dagger$ ). הגרפים של  $preception$  לעומת  $svm$ . ניתן ממש לראות איך המישורים מתאחדים:



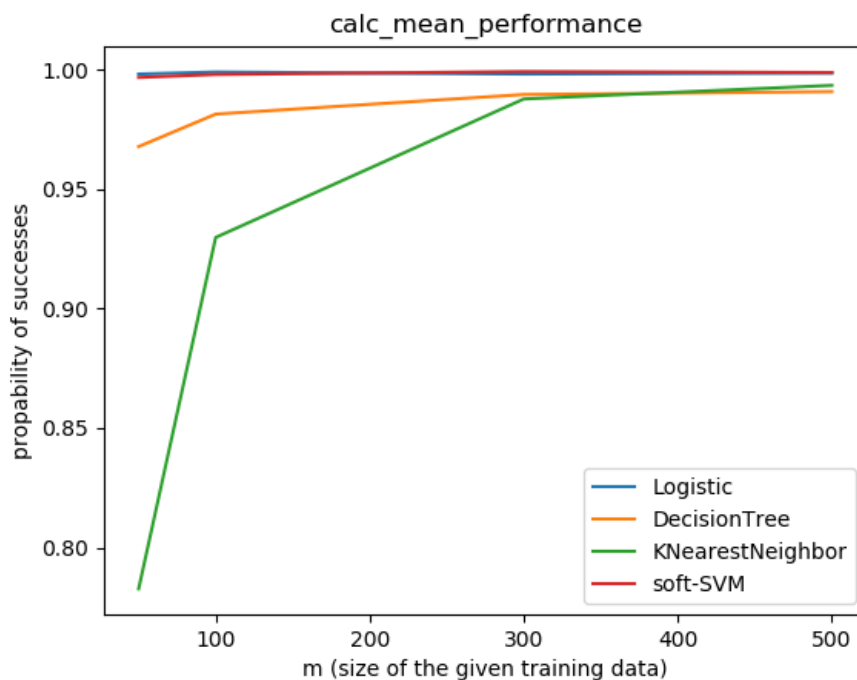
## 7.1 ההשוואה בין בשיטות השונות ל LDA.

קל לראות ש LDA לא נותן לנו שיערוך טוב. (כלומר כן טוב אבל פחות) אני מאמין שהוא סובל יותר מאובר פיט. מאחר ומראש הוא לא בהכרח מייצג מישור ולכן ספיציפית לבעייה הזאת הוא פחות מתאים.



## 8 זיהוי ספרות :

ההשוואה על זיהוי הספרות, אני חייב להגיש שממש הואשמתי מי מידת הדיוק. שיטת ה SVM לקחה את רוב זמן החישוב כאשר על הקלטים הגדולים זמן הריצה היה באזור  $0.05 [sec]$ . כל שאר השיטות ירדו מ 0.025 שניות עבור הקלטים הגדולים.



## 2 ./comparsion.py

```
1  import numpy as np
2  import pandas as pd
3  from sklearn.svm import SVC
4
5  from sklearn.linear_model import LogisticRegression
6  from sklearn.tree import DecisionTreeClassifier
7  from sklearn.neighbors import KNeighborsClassifier
8
9  import matplotlib.pyplot as plt
10 from mpl_toolkits.mplot3d import Axes3D # <--- This is important for 3d plotting
11 import pandas as pd
12 from pandas import DataFrame
13 from plotnine import *
14
15 import matplotlib as mpl
16 import matplotlib.pyplot as plt
17
18 from random import random
19
20 from models import SVM, LDA, Perceptron
21
22
23 def label_f(weight, bias):
24     def _label_f(x):
25         return np.sign( np.dot(weight, x) + bias)
26     return _label_f
27
28 def draw_points(m, label_function=label_f(np.array([0.3, -0.5]), 0.1)):
29     def _draw_points_n(m,n):
30         return np.random.multivariate_normal(
31             np.zeros(n), np.identity(n), m)
32
33     X = _draw_points_n(m, 2)
34     y = np.array([ label_function(vec) for vec in X ])
35     return X,y
36
37 def analyze_clsifiers( ):
38
39     # def generate_line_prec(prec):
40
41
42     def plot(prec, _svm, X, y):
43         plt.scatter( X[y == -1][:,0], X[y == -1][:,1])
44         plt.scatter( X[y == 1][:,0], X[y == 1][:,1])
45
46         min_x, max_x = min(X[:,0]), max(X[:,0])
47         min_y, max_y = min(X[:,1]), max(X[:,1])
48         print(min_x, max_x)
49
50         _min_range, _max_range = min(min_x, min_y) , max(max_x, max_y)
51         print(prec.W)
52         xx = [_min_range, _max_range]
53
54         def get_y(W, _x):
55             return -(W[0] + _x * W[1])/ W[2] if W[2] != 0 else -W[0]
56
57         def get_y_prep(_x):
58             return get_y(prec.W, _x )
59
```

```

60     def get_y_svm(_x):
61         print(_svm.coef_()[0])
62         return get_y(_svm.coef_()[0], _x)
63
64     def get_true_y(_x):
65         return 0.1/0.5 + 0.3/0.5 * _x
66
67     plt.xlim([_min_range, _max_range])
68     plt.ylim([_min_range, _max_range])
69
70
71
72     middle = (_min_range + _max_range) / 2
73
74     def print_line(msg, _f, _color):
75         xx = [_min_range, _max_range]
76         yy = [_f(x) for x in xx]
77         _x = middle + 2 * (0.5 - random())
78         _y = _f(_x)
79         plt.plot(xx, yy, color=_color)
80         plt.annotate(msg, color=_color,
81                      xy=(_x, _y), xycoords='data',
82                      xytext=(_x + 0.3, _y), textcoords='data',
83                      arrowprops=dict(arrowstyle="->"))
84
85     print_line("prep", get_y_prep, "C5")
86     print_line("svm", get_y_svm, "C4")
87     print_line("true plane", get_true_y, "C2")
88     plt.title("svm vs prep")
89     plt.xlabel("x")
90     plt.ylabel("y")
91     plt.show()
92
93     for m in [5, 10, 15, 25, 70]:
94         X, y = draw_points(m)
95         blues, reds = X[y==1], X[y==0]
96         _modes = [Perceptron(), SVM()]
97         for _model in _modes:
98             _model.fit(deepcopy(X), y)
99
100         plot(_modes[0], _modes[1], X, y)
101
102
103     def expanded_analyze_clssifiers():
104         times, k = 500, 1000
105         modles = []
106         models_num = 3
107
108
109     def genrate_real_plane(m):
110         _f = label_f(np.array([random(), random()]), random())
111         X, y = draw_points(m, label_function=_f)
112         while (-1 not in y) or (1 not in y):
113             X, y = draw_points(m, label_function=_f)
114         return X, y, _f
115
116     def accur(mod, _f, Z):
117         _prob = 0
118         for x, y in zip(map(_f, Z), mod.predict(Z)):
119             if x == y:
120                 _prob += 1
121         return _prob/len(Z)
122
123     def one_iteraion(m):
124         _modes = [Perceptron(), SVM(), LDA()]
125         X, y, _f = genrate_real_plane(m)
126         ret = []
127         for _model in _modes:

```

```

128         _model.fit(deepcopy(X),y)
129         Z, _ = draw_points(k)
130         ret.append( accur(_model, _f, Z) )
131     return np.array(ret)
132
133 def calc_mean_performance(M = [5, 10, 15, 25, 70] ):
134     ret = []
135     for m in M:
136         _mean = np.zeros(models_num)
137         for _ in range(times):
138             _mean += one_iteraion(m)
139         ret.append( _mean/ times )
140     return M, np.array( ret )
141
142 m, mean_performance = calc_mean_performance()
143 for _model_num, _name in enumerate(["perc", "svm", "lda"]):
144     print(mean_performance)
145     plt.plot( m , mean_performance[:,_model_num] )
146 plt.legend( ["perc", "svm", "lda"] )
147 plt.title("calc_mean_performance")
148 plt.xlabel("m (size of the given training data)")
149 plt.ylabel("propability of successes")
150 plt.show()
151
152 if __name__ == "__main__" :
153     X,y = draw_points(10)
154     #p = Perceptron()
155
156
157     from copy import deepcopy
158
159     models_class = [ Perceptron, SVM, Logistic, DecisionTree, LDA ]
160     models = [ ]
161     for mod in models_class:
162         models.append( mod( ) )
163         print("{} init ".format( type(mod) ))
164
165     for mod in models:
166         mod.fit(deepcopy(X),y)
167         print("{} fit ".format( type(mod) ))
168
169     for mod in models:
170         print(mod.predict(deepcopy(X)))
171         print("{} predict ".format( type(mod) ))
172
173     analyze_clssifiers()
174     expanded_analyze_clssifiers()

```



## 3 ./mnsit data.py

```
1  import pandas as pd
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import tensorflow as tf
5
6  from models import Logistic, DecisionTree, KNearestNeighbor, SVM
7  from copy import deepcopy
8
9  import time
10
11 def load_data():
12     mnist = tf.keras.datasets.mnist
13     (x_train, y_train), (x_test, y_test) = mnist.load_data()
14     train_images = np.logical_or((y_train == 0), (y_train == 1))
15     test_images = np.logical_or((y_test == 0), (y_test == 1))
16     x_train, y_train = x_train[train_images], y_train[train_images]
17     x_test, y_test = x_test[test_images], y_test[test_images]
18
19     return x_train, y_train, x_test, y_test
20
21
22 def rearrange_data(X):
23     return np.array( [ x.flatten() for x in X ])
24
25
26 def expanded_analyze_clssifiers(x_train, y_train, x_test, y_test):
27     times, k = 50, 100
28     modles = []
29     models_num = 4
30
31     def generate( m, X, y ):
32         indexs = [False] * len(X)
33         _indexs = np.random.choice( list(range(len(X)) ), size=m)
34         for _index in _indexs:
35             indexs[_index] = True
36
37         return X[indexs], y[indexs], indexs
38
39     def accur(mod, indexs , Z):
40         _prob = 0
41         for x,y in zip(y_test[indexs], mod.predict(Z)):
42             if x == y:
43                 _prob +=1
44         return _prob/len(Z)
45
46     def one_iteraion(m):
47         _modes = [Logistic(), DecisionTree(), KNearestNeighbor(), SVM()]
48         X, y, indexs = generate(m, x_train, y_train)
49
50         # your code
51
52
53         while (0 not in y) or (1 not in y) :
54             X, y, indexs = generate(m, x_train, y_train)
55         ret = []
56         for _model in _modes:
57             start_time = time.time()
58             _model.fit(deepcopy(X),y)
59             elapsed_time = time.time() - start_time
```

```

60         print("train : {} takes {}".format(_model, elapsed_time))
61         Z, _, indexes = generate(k, x_test, y_test)
62         ret.append( accur(_model, indexes, Z) )
63     return np.array(ret)
64
65 def calc_mean_performance(M = [50, 100, 300, 500] ):
66     ret = []
67     for m in M:
68         _mean = np.zeros(models_num)
69         for _ in range(times):
70
71             _mean += one_iteraion(m)
72         ret.append( _mean/ times )
73     return M, np.array( ret )
74
75 m, mean_performance = calc_mean_performance()
76 for _model_num, _name in enumerate(["Logistic", "DecisionTree", "KNearestNeighbor", "soft-SVM"]):
77     plt.plot( m , mean_performance[:,_model_num] )
78     plt.legend( ["Logistic", "DecisionTree", "KNearestNeighbor", "soft-SVM"] )
79     plt.title("calc_mean_performance")
80     plt.xlabel("m (size of the given training data)")
81     plt.ylabel("propability of successes")
82     plt.show()
83
84
85 if __name__ == '__main__':
86     x_train, y_train, x_test, y_test = load_data()
87     expanded_analyze_clssifiers(rearrange_data(x_train),
88                                 rearrange_data(y_train), rearrange_data(x_test), rearrange_data(y_test))

```

## 4 ./models.py

```
1  import numpy as np
2  import pandas as pd
3  from sklearn.svm import SVC
4
5  from sklearn.linear_model import LogisticRegression
6  from sklearn.tree import DecisionTreeClassifier
7  from sklearn.neighbors import KNeighborsClassifier
8
9  import matplotlib.pyplot as plt
10 from mpl_toolkits.mplot3d import Axes3D # <--- This is important for 3d plotting
11 import pandas as pd
12 from pandas import DataFrame
13 from plotnine import *
14
15 import matplotlib as mpl
16 import matplotlib.pyplot as plt
17
18 from random import random
19
20 class abcModel:
21
22     def __init__(self):
23         self.mod = None
24
25     def fit(self, X, y):
26         self.mod.fit(self.bias(X), y )
27
28     def predict(self, X):
29         return self.mod.predict(self.bias(X))
30
31     def get_hyperplan(self):
32         pass
33
34     def bias(self, X):
35         return np.insert(X,0, 1, 1)
36
37     def draw(self):
38         pass
39
40     def score(self, X, y):
41         return {
42             num_samples : 0,
43             error : 0,
44             accuracy :0,
45             FPR : 0,
46             TPR : 0,
47             precision : 0,
48             recall : 0
49         }
50
51 # [v]
52 class Perceptron(abcModel):
53     def __init__(self):
54         self.W = np.array( [] )
55         super().__init__()
56
57     def fit(self, _X, y):
58         X = self.bias(_X)
```

```

60     self.W = np.zeros( shape = X.shape[1] )
61     def not_classifiy():
62         return [ (x,y) for x in X if self.predict(x) != y ]
63
64     _updated = True
65     while _updated:
66         _updated = False
67         for i in range(len(y)):
68             if np.dot(self.W, X[i]) * y[i] <= 0:
69                 self.W = self.W + (X[i]* y[i])
70                 _updated = True
71     return self.W
72
73     def predict(self, X):
74         return np.sign(self.W @ self.bias(X).transpose())
75
76 class LDA(abcModel):
77     def __init__(self):
78         super().__init__()
79
80
81     def fit (self, X, y):
82         X = self.bias(X)
83
84         def gen_delta_y(X ,y ,y_val):
85             _X = X[ y == y_val ]
86             lnP = np.log( len( y == y_val ) / len(y) )
87
88             aritmetic_mean = np.array( [np.mean( u ) for u in _X.transpose()] )
89             _cov = np.cov( X.transpose() )
90             _inv_cov = np.linalg.pinv(_cov)
91
92             def delta(x):
93                 return x.transpose() @ _inv_cov @ aritmetic_mean -\
94                     0.5*aritmetic_mean.transpose() @ _inv_cov @ aritmetic_mean +\
95                     lnP
96
97             return delta
98
99         self.deltas = [ gen_delta_y(X ,y ,y_val) for y_val in [-1, 1] ]
100
101     def predict(self, U):
102         return np.array([ { 0 : -1.0, 1 : 1.0 }[np.argmax( [_delta(u) for _delta in self.deltas] ) ] for u in self.bias(U) ])
103
104
105 class SVM(abcModel):
106     def __init__(self):
107         super().__init__()
108         self.mod = SVC(C=1e10, kernel='linear')
109
110     def coef_(self):
111         return self.mod.coef_
112
113 class Logistic(abcModel):
114     def __init__(self):
115         super().__init__()
116         self.mod = LogisticRegression(solver='liblinear')
117
118 class DecisionTree(abcModel):
119     def __init__(self):
120         super().__init__()
121         self.mod = DecisionTreeClassifier()
122
123 class KNearestNeighbor(abcModel):
124     def __init__(self):
125         super().__init__()

```

```

128         self.mod = KNeighborsClassifier(n_neighbors=40)
129
130
131
132     def label_f(weight, bias):
133         def _label_f(x):
134             return np.sign( np.dot(weight, x) + bias)
135         return _label_f
136
137     def draw_points(m, label_function=label_f(np.array([0.3, -0.5]), 0.1)):
138         def _draw_points_n(m,n):
139             return np.random.multivariate_normal(
140                 np.zeros(n), np.identity(n), m)
141
142         X = _draw_points_n(m, 2)
143         y = np.array([ label_function(vec) for vec in X ])
144         return X,y
145
146     def analyze_clssifiers( ):
147
148
149         # def generate_line_prec(prec):
150
151         def plot(prec, _svm, X, y):
152             plt.scatter( X[y == -1][:,0], X[y == -1][:,1])
153             plt.scatter( X[y == 1][:,0], X[y == 1][:,1])
154
155             min_x, max_x = min(X[:,0]), max(X[:,0])
156             min_y, max_y = min(X[:,1]), max(X[:,1])
157
158             _min_range, _max_range = min(min_x, min_y) , max(max_x, max_y)
159             xx = [_min_range, _max_range]
160
161             def get_y(W, _x):
162                 return -(W[0] + _x * W[1])/ W[2] if W[2] != 0 else -W[0]
163
164             def get_y_prep(_x):
165                 return get_y(prec.W, _x )
166
167             def get_y_svm(_x):
168                 print(_svm.coef_()[0])
169                 return get_y(_svm.coef_()[0], _x)
170
171             def get_true_y(_x):
172                 return 0.1/0.5 + 0.3/0.5 * _x
173
174             plt.xlim([_min_range,_max_range])
175             plt.ylim([_min_range,_max_range])
176
177
178             middle = (_min_range + _max_range) /2
179
180
181             def print_line(msg, _f, _color):
182                 xx = [_min_range , _max_range ]
183                 yy = [ _f(_x) for _x in xx ]
184                 _x = middle + 2 * (0.5 - random())
185                 _y =_f(_x)
186                 plt.plot(xx, yy, color=_color )
187                 plt.annotate(msg, color=_color,
188                     xy=( _x, _y), xycoords='data',
189                     xytext=(_x + 0.3 , _y), textcoords='data',
190                     arrowprops=dict(arrowstyle="->"))
191
192             print_line("prep", get_y_prep, "C5")
193             print_line("svm" , get_y_svm , "C4")
194             print_line("true plane" , get_true_y ,"C2")
195             plt.title("svm vs prep")

```

```

196         plt.xlabel("x")
197         plt.ylabel("y")
198         plt.show()
199
200     for m in [5, 10, 15, 25, 70]:
201         X, y = draw_points(m)
202         blues, reds = X[y==1], X[y==0]
203         _modes = [Perceptron(), SVM()]
204         for _model in _modes:
205             _model.fit(deepcopy(X), y)
206
207         plot( _modes[0], _modes[1], X, y)
208
209
210 def expanded_analyze_clssifiers():
211     times, k = 7, 1000
212     modles = []
213     models_num = 3
214
215
216     def genrate_real_plane(m):
217         _f = label_f(np.array([random(), random()]), random())
218         X, y = draw_points(m, label_function=_f)
219         while (-1 not in y) or (1 not in y):
220             X, y = draw_points(m, label_function=_f)
221         return X, y, _f
222
223     def accur(mod, _f, Z):
224         _prob = 0
225         for x, y in zip(map(_f, Z), mod.predict(Z)):
226             if x == y:
227                 _prob += 1
228         return _prob/len(Z)
229
230     def one_iteraion(m):
231         _modes = [Perceptron(), SVM(), LDA()]
232         X, y, _f = genrate_real_plane(m)
233         ret = []
234         for _model in _modes:
235             print(type(_model))
236             _model.fit(deepcopy(X), y)
237             Z, _ = draw_points(k)
238             ret.append( accur(_model, _f, Z) )
239         return np.array(ret)
240
241     def calc_mean_performance(M = [5, 10, 15, 25, 70] ):
242         ret = []
243         for m in M:
244             _mean = np.zeros(models_num)
245             for _ in range(times):
246                 _mean += one_iteraion(m)
247             ret.append( _mean/ times )
248         return M, np.array( ret )
249
250     m, mean_performance = calc_mean_performance()
251     for _model_num, _name in enumerate(["perc", "svm", "lda"]):
252         print(_name)
253         print(mean_performance)
254         plt.plot( m , mean_performance[:, _model_num] )
255     plt.legend( ["perc", "svm", "lda"] )
256     plt.title("calc_mean_performance")
257     plt.xlabel("m (size of the given training data)")
258     plt.ylabel("propability of successes")
259     plt.show()
260
261 if __name__ == "__main__" :
262     X, y = draw_points(10)
263     #p = Perceptron()

```

```

264
265
266     from copy import deepcopy
267
268     models_class = [ Perceptron, SVM, Logistic, DecisionTree, LDA ]
269     models = [ ]
270     for mod in models_class:
271         models.append( mod( ) )
272         print("{} init ".format( type(mod) ))
273
274     for mod in models:
275         mod.fit(deepcopy(X),y)
276         print("{} fit ".format( type(mod) ))
277
278     for mod in models:
279         print(mod.predict(deepcopy(X)))
280         print("{} predict ".format( type(mod) ))
281
282     #analyze_clssifiers()
283     expanded_analyze_clssifiers()

```