

Sensor Drift Dataset

黃泰揚 00857114

問題描述：

前陣子研究機器學習的相關論文，發現有個很經典的機器學習訓練問題叫”感測器飄移 (sensor drift)”，發生原因是由於感測器會因老化、污染、甚至移位，此專題的目標是利用訓練的結果辨認該問題；同時根據課程所學來進行各種延伸探討。

為此，我準備了一個出現 sensor drift 問題的資料集：

<https://archive.ics.uci.edu/ml/datasets/Gas+Sensor+Array+Drift+Dataset+at+Different+Concentrations>

這個資料集包含由 16 個化學感測器對 6 種不同氣體(丙酮、乙醛、乙醇、乙烯、阿摩尼亞、甲苯)於不同濃度下，從 2008/1 至 2011/2，歷經 36 個月所取得的 13910 樣本。每個樣本有 128 特徵(16x8)。這個問題是根據此 128 特徵，判斷氣體是那 6 種氣體裡的哪一種氣體。此資料集的樣本依照收集的時間順序安排，並分成 10 個資料檔，batch1.dat、...、batch10.dat。其中 batch2~10 應該會出現誤差越來越大的趨勢。

如果沒注意到 sensor drift 發生了，就直接把 batch1~10 合成一個資料集做訓練，那後果可想而知(underfitting 或訓練出來了但在實際預測時準確率很低)，而避免發生 sensor drift 的方法其實不少，定期校準感測器/定期重新訓練分類器/只使用出現問題的資料做訓練都可以有效解決(實際上還有一種專門處理該問題的方法叫做 domain adaptation，但是太難了我不會)。

由於理論上 batch1 的資料集誤差是最少的，所以我利用 batch1 訓練多個分類器模型，並拿其他 batch 來做預測，看是否可以根據這個結果判斷是否出現感測器飄移的問題。

1.練習使用 scikit-learn 中的 pipeline 功能建置模型：

為了使 batch1 的訓練誤差盡量小，使用 pipeline+K-fold 來選取最佳超參數這裡根據期中考前所教的內容建置了 4 種模型：

svm_linear0: 假設該問題線性可分(實際上應該不可)，來找 sv 的模型。

dt: 用決策樹來建置的模型。

svm_linear: 在進行線性 svm 前，使用資料分群，先進行 stdscaler(標準化特徵)，以下三種分類器都有使用。

svm_rbf: 找到的 sv 不限定為直線。

knn:找前 k 個最接近的訓練樣本演算法(這裡的 k=[3,5,7])

*svm 系列的超參數 C(對誤差容忍度，越高找到的線會越崎嶇)=[0.01,0.1,1,10,100],其中 rbf 的超參數 gamma(越大找到的 sv 越少)=[0.001,0.01,0.1,1]

```
from sklearn import datasets, svm, model_selection
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# StratifiedFold splits=5, shuffle=True, random_state=3
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=3)

# defining dictionaries: pipeline_knn_param0, pipeline_rbf_svm_param0, pipeline_linear_svm_param, linear_svm_param0
pipeline_knn_param0 = {'knn__n_neighbors': [3, 5, 7]}
pipeline_rbf_svm_param0 = {'svc__C': [0.01, 0.1, 1, 10, 100], 'svc__gamma': [0.001, 0.01, 0.1, 1]}
pipeline_linear_svm_param = {'svc__C': [0.01, 0.1, 1, 10, 100], 'fr__k': [50, 75]}
linear_svm_param0 = {'C': [0.01, 0.1, 1, 10, 100]}

# defining classifiers: knn, svm_rbf, svm_linear, svm_linear0, dt
svm_linear0 = GridSearchCV(estimator=svm.SVC(kernel='linear', gamma=1, C=1, probability=True, decision_function_shape='ovo'), param_grid=linear_svm_param0, cv=kf)
dt = DecisionTreeClassifier()
knn = GridSearchCV(estimator=Pipeline([('fr', SelectKBest(score_func=mutual_info_classif, k=50)), ('stdscaler', StandardScaler()), ('knn', neighbors.KNeighborsClassifier)]), param_grid=pipeline_knn_param0, cv=kf)
svm_rbf = GridSearchCV(estimator=Pipeline([('fr', SelectKBest(score_func=mutual_info_classif, k=50)), ('stdscaler', StandardScaler()), ('svc', svm.SVC(kernel='rbf', gamma=1, C=1, probability=True, decision_function_shape='ovo'))]), param_grid=pipeline_rbf_svm_param0, cv=kf)
svm_linear = GridSearchCV(estimator=Pipeline([('fr', SelectKBest(score_func=mutual_info_classif, k=50)), ('stdscaler', StandardScaler()), ('svc', svm.SVC(kernel='linear', gamma=1, C=1, probability=True, decision_function_shape='ovo'))]), param_grid=pipeline_linear_svm_param, cv=kf)

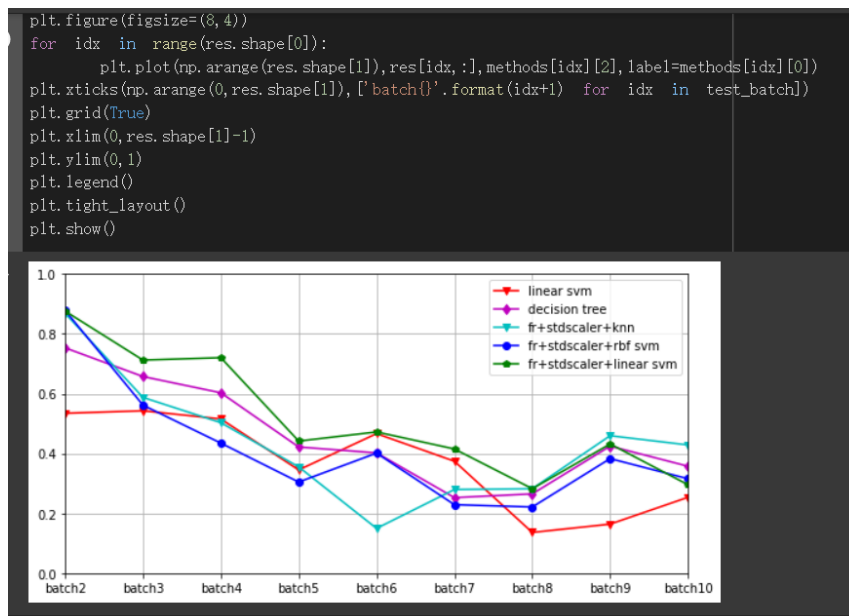
# input model
methods = [('linear', svm_linear0, 'rr-'),
            ('decision tree', dt, 'dt-'),
            ('fr+stdscaler+knn', knn, 'cv-'),
            ('fr+stdscaler+rbf svm', svm_rbf, 'bo-'),
            ('fr+stdscaler+linear svm', svm_linear, 'sp-')]

# prepare the training and test sets
train_batch = [0]
test_batch = [x for x in range(1, len(dataset)) if x not in train_batch]
```

2. 將結果可視化:

可以看出隨著時間推移，各模型的準確度開始逐漸下降，不過由於我並沒有無誤差的數據資料，所以無法確定何種模型的辨認效果一定較好，但大部分都是可以清楚辨認的(不過 linear svm0 模型看起來似乎有 underfitting 的情形)。

然後比較意外的結果是資料分群對於 svm linear 的提升效果非常顯著，甚至高過分類能力較高的 svm rbf。



3. 練習建構多層 FNN:

四個 FNN 的神經架構如下：

model1: 128->BN->512-512->Dropout(0.1)->6

model2: 128->128->128->128->6

model3: 128->BN->512->128->64->6

model4: 128->32->32->32->32->Dropout(0.1)->6

其中第一個數字表示輸入向量的維數，最後一個數字表示輸出單元的數量，另一個數字表示隱藏層的單元數，BN 表示批量歸一化層，Dropout(0.1) 表示 dropout 層，dropout ratio 設置為 0.1。隱藏單元的激活函數是 relu，輸出單元的激活函數是 softmax。損失函數是 sparse categorical cross entropy，優化器是 adam。

```
def def_model1():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.InputLayer(input_shape=(128,)))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Dense(512, activation='relu'))
    model.add(tf.keras.layers.Dense(512, activation='relu'))
    model.add(tf.keras.layers.Dropout(0.1))
    model.add(tf.keras.layers.Dense(6, activation='softmax'))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

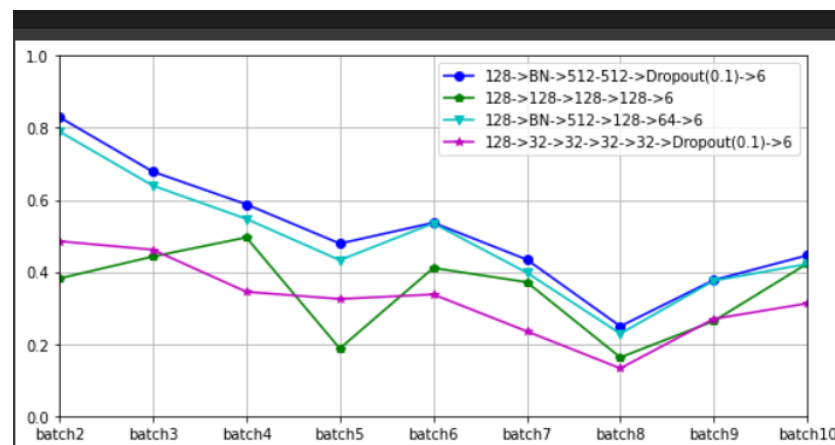
def def_model2():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.InputLayer(input_shape=(128,)))
    model.add(tf.keras.layers.Dense(128, activation='relu'))
    model.add(tf.keras.layers.Dense(128, activation='relu'))
    model.add(tf.keras.layers.Dense(128, activation='relu'))
    model.add(tf.keras.layers.Dense(6, activation='softmax'))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

def def_model3():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.InputLayer(input_shape=(128,)))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Dense(512, activation='relu'))
    model.add(tf.keras.layers.Dense(128, activation='relu'))
    model.add(tf.keras.layers.Dense(64, activation='relu'))
    model.add(tf.keras.layers.Dense(6, activation='softmax'))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

4. 和單一分類器的比較訓練結果:

由於 FNN 理論上對於此問題為較精準的分類器(因為此資料集的 Feature 很多，所以透過刪減不必要的資訊應該可以大幅增加學習的效率)，所以推論藍線的兩種模型應該更接近實際的誤差值。

但實驗結果顯示 svm linear + 資料集群，似乎是這個問題的最佳解。



5. 使用自定義模型 myGaussianClassifier:

使用第三章的內容來製作高斯分類器，假設 $P(X|C=i), \forall i$ $P(X|C=i), \forall i$ 呈現高斯分佈 $N(\mu_i, \Sigma_i)$ ，看一下高斯分類器的分類能力如何。

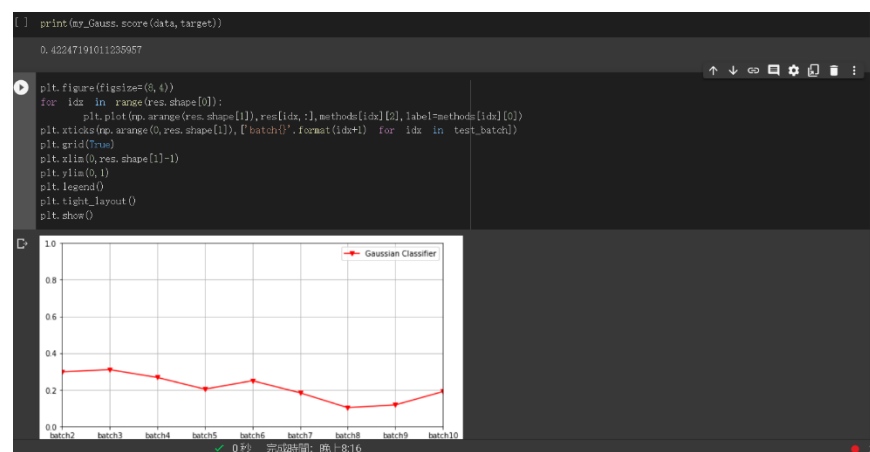
```
from scipy.stats import multivariate_normal
from sklearn.metrics import accuracy_score

class myGaussianClassifier(GaussianEstimator, ClassifierMixin):
    def __init__(self, alpha=1.e-5):
        if isinstance(self, myGaussianClassifier):
            super(myGaussianClassifier, self).__init__()
            self.alpha = alpha

    def fit(self, train, target):
        N, d = train.shape
        label = np.sort(np.unique(target.ravel()))
        self.c_ = label.size
        self.d_ = d
        self.prior_ = np.zeros((self.c_,))
        self.mean_ = np.zeros((self.c_, self.d_))
        self.cov_ = np.zeros((self.c_, self.d_, self.d_))
        # 計算 mean, covariance
        for cid, y in enumerate(label):
            idx = np.nonzero(target.ravel()==y)
            self.cov_[cid] = np.cov(train[idx], rowvar=False)*self.alpha*np.eye(d)
            self.mean_[cid] = np.average(train[target==cid], axis=0)
            self.prior_[cid] = np.sum(target.ravel()==cid)/target.size
        self.P0_ = multivariate_normal(self.mean_[0], self.cov_[0], allow_singular=True)
        self.P1_ = multivariate_normal(self.mean_[1], self.cov_[1], allow_singular=True)
        return self

    def predict(self, X, y=None):
        preds = []
        for x in X:
            if self.P0_.pdf(x)*self.prior_[0]>self.P1_.pdf(x)*self.prior_[1]:
```

可以看出高斯分類器由於模型過於簡單，對於複雜的問題效果很差。
10~30%的準確度和用猜的差不多($1/6=0.166$)



6. 判斷模型相似度:

這裡使用看起來訓練效果最好的 `fr+stdscaler+linear svm` 來當作基準(不比較 FNN 是由於該系列不支援 `cross_val_score`)

根據顯著水準(significance level)來定義:

以 paired t-test 顯著程度 0.05 前提下，平均準確率最高那個分類器與另外 6 個分類器，判斷在平均準確率上是否有顯著差異。

```
for i in range(len(methods_score)):
    method_t, method_pvalue = stats.ttest_1samp (methods_score[4]-methods_score[i], 0)
    print("{} pvalue = {}".format(methods[i][0], method_pvalue))

linear svm pvalue = 0.10870095132492331
decision tree pvalue = 0.0777416409478997
fr+stdscaler+knn pvalue = 1.0
fr+stdscaler+rbf svm pvalue = 0.20799999999999935
fr+stdscaler+linear svm pvalue = nan
Gaussian Classifier pvalue = 1.5343629751998116e-07
```

可以看到一個很有趣的結果，居然只有 Gaussian Classifier 不具有顯著差異，如果只從前面幾個的數據來判斷，和 fr+stdscaler+linear svm 最像的分類器應該是 decision tree，實際上前面 5 個分類器的結果也和前面的實驗吻合，但前面的實驗忽略了一個很重要的數據:batch 1 本身的訓練成果，我認為這可能是主要原因。

結果討論&實驗心得：

1. 這個問題其實很容易發現:可以看出即使不使用多層的神經網路，基本的機器學習技術也足以判斷是否發生了感測器位移。
2. 沒有完美的模型:每種模型都有他是和分類的問題，很多時候模型複雜不一定是好的，像在此問題當中 linear svm + stdscale 就是相對較好的策略，但它並不是最複雜/跑最快/參數最多的。
3. 多層神經網路的建置訣竅:由低維->高維->低維通常是相對較好的策略。
4. 機器學習上的”相似”和直觀的”相似”差很多:如果完全不看分類結果，我會認為和 linear svm + stdscale 最像的會是 linear rbf + stdscale 或是直接 linear svm 模型，但就算看了分類結果，我也不會想到 gaussian classifier 會是最像的，只有實際算出來才知道。

再研究機器學習相關論文時，偶然看到的這個問題給了我很大的啟發，有時候資料訓練不起來不一定是訓練模型本身的問題(overfitting/underfitting)，外在因素也要考量，因為機器學習的課程對測試資料的正確性這部份很少著墨，所以我藉由這專題特別去研究。