Artificial Intelligence

PBL problem Ⅱ

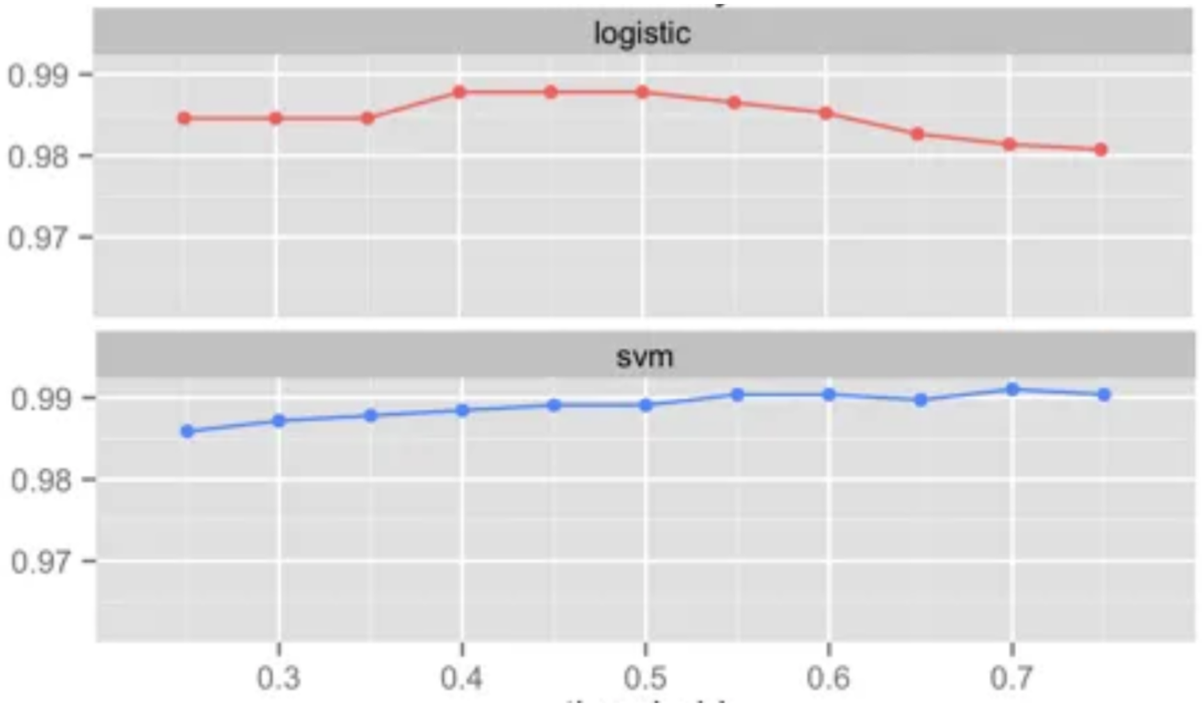
SGD-SVM

Team. 6
2015038195 김재희
2015040900 박영빈
2015050746 유승완
2017011676 김나연
2017012142 안다영
2017012479 이하영

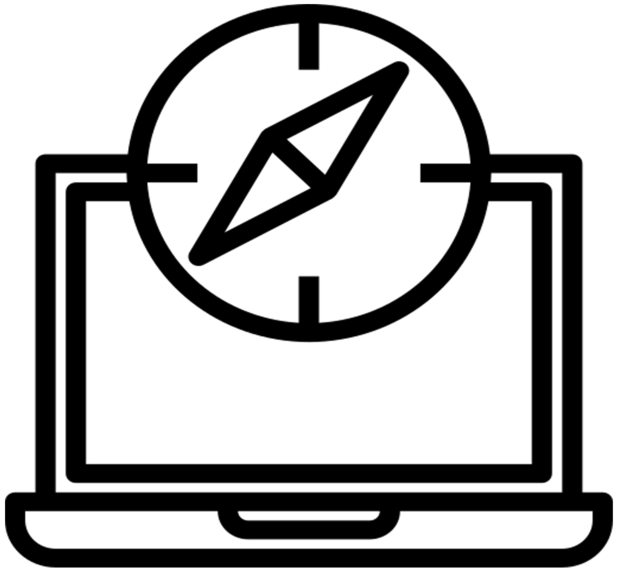
INDEX

Ⅰ. Problem

accuracy



Threshold

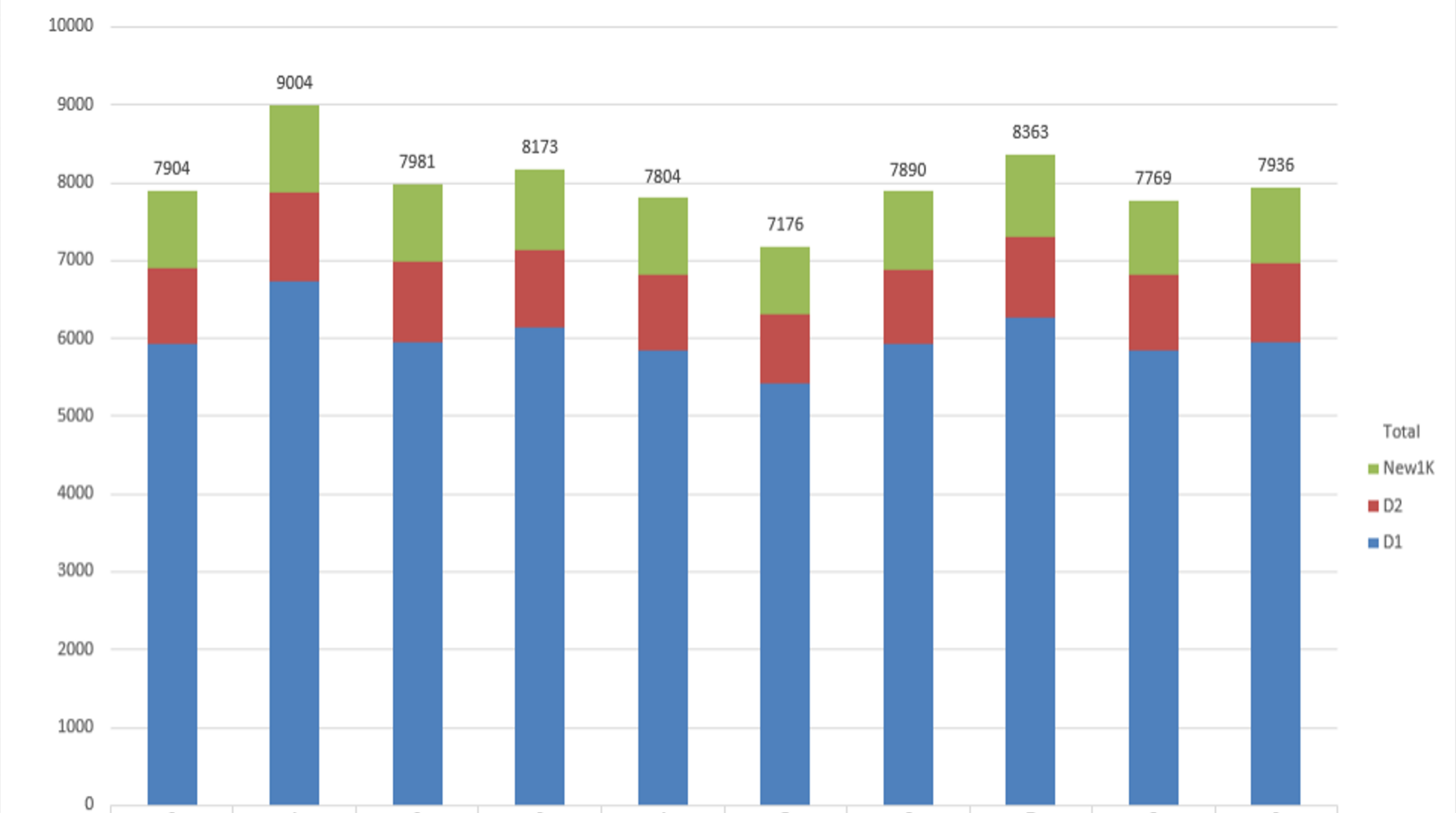


Taking too much time!

Graph Source: http://www.win-vector.com/blog/2015/02/does-balancing-classes-improve-classifier-performance/

Ⅱ. Data

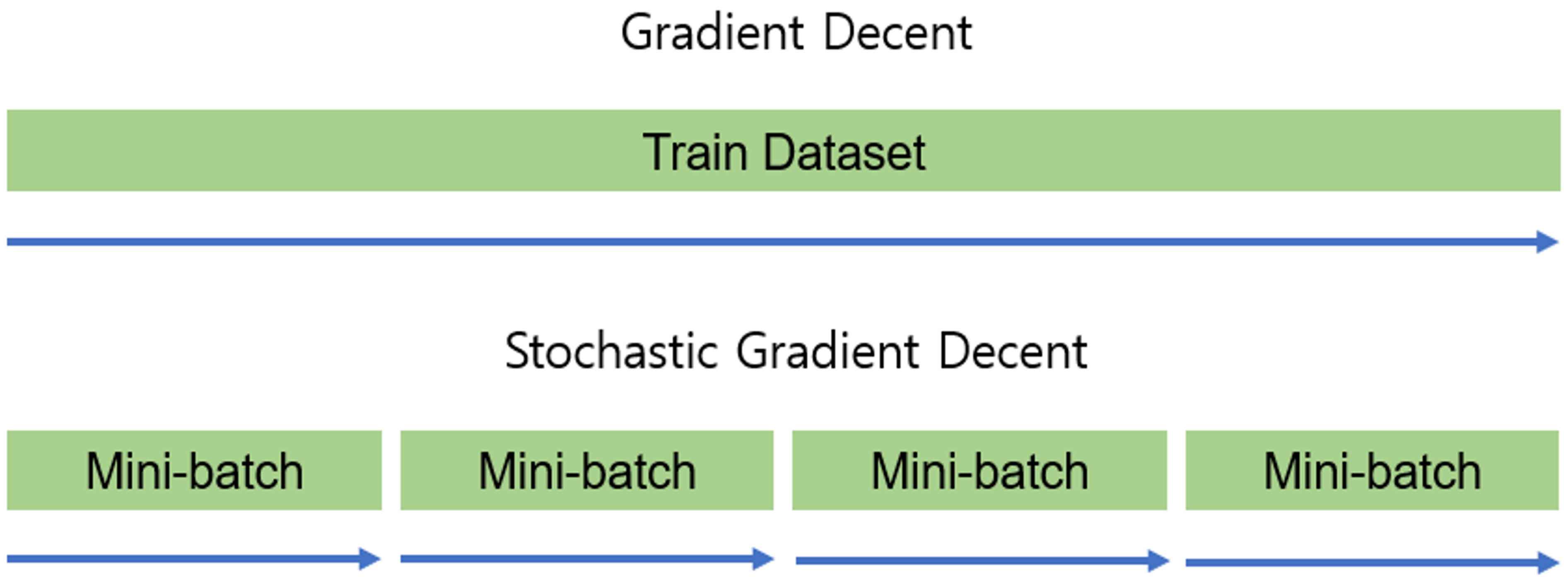
D1 + D2 + New 1K



	0	1	2	3	4	5	6	7	8	9
Total	7904	9004	7981	8173	7804	7176	7890	8363	7769	7936
New1K	1001	1127	991	1032	980	863	1014	1070	944	978
D2	980	1135	1032	1010	982	892	958	1028	974	1009
D1	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949

- The company prepared 'New 1k' image dataset
- D1 + D2 + New 1k datasets will be used to train our model
- 'Other New 5k dataset' and 'D2 dataset' will be D3 to test our model

Ⅱ. Data



- We will validate our model using Stratified Kfold with GridSearchCV
- Mini batch size, learning rate will be an important parameter in SGD

Ⅱ. Data

Preprocessing - One-hot encoding and Scaling

```
def one_hot(self, y):
    """
    one-hot encode : 1 is hot, -1 is cold
    """
    onehot = np.ones(self.n_classes)

    for i in range(self.n_classes):
        if y != i:
            onehot[i] = -1

    return onehot
```

```
print(D1_labels_train[0])
```

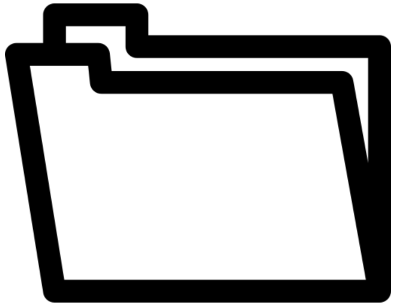
```
5
```

```
one_hot(D1_labels_train[0])
```

```
array([-1., -1., -1., -1., -1.,  1., -1., -1., -1., -1.])
```

sklearn.preprocessing.StandardScaler()

Ⅲ. SGD-SVM



- team6/
  - __init__.py
  - team6.py : read_idx(), main()
  - utils.py : preprocess(), deskew() etc.
  - SVM.py : SGDSVM - fit(), predict(), _SGD() etc.
  - dataset
    - train dataset
    - test dataset

Ⅲ. SGD-SVM

SGDSVM - __init__()

```
def __init__(self, C=1000.0, eta=0.1, max_iter=5, batch_size=128, random_state=1234):

    """
        - C : Penalty parameter C of the error term.
        - eta : Learning Rate.
        - max_iter : Hard limit on iterations within solver, or -1 for no limit.
        - batch_size : Mini batch size for SGD(Stochastic Gradient Descent).
        - random_state : Random seed number

    """
```

Ⅲ. SGD-SVM

SGDSVM - fit() Init Parameter

```
# init parameter
self.rgen = np.random.RandomState(self.random_state)
W = self.rgen.normal(loc=0.0, scale=0.01, size=(self.n_features, self.n_classes))
b = np.ones((1, self.n_classes))
W_ = W[:,:]
b_ = b[:,:]

# the best W and b that have the best accuracy
self.best_W = W_[:]
self.best_b = b_[:]
```

Ⅲ. SGD-SVM

SGDSVM - fit() Mini-batch without Replacement

```
ss = np.arange(n) # range of the number of data points
n_batches = int(np.ceil(len(y) / self.batch_size)) # number of batches

# random sampling without replacement
self.rgen.shuffle(ss)

for i in range(n_batches):
    # mini-batch
    batch_idx = ss[self.batch_size * i: self.batch_size * (i + 1)]
```

Ⅲ. SGD-SVM

SGDSVM - fit() Update with Weight Averaging

$$
\begin{aligned}
w_{k+1} &\leftarrow w_k - \eta \nabla_w \tilde{F}(w_k, b_k) \\
b_{k+1} &\leftarrow b_k - \eta \nabla_b \tilde{F}(w_k, b_k) \\
\bar{w}_{k+1} &\leftarrow \frac{k}{k+1}\bar{w}_k + \frac{1}{k+1} w_{k+1} \\
\bar{b}_{k+1} &\leftarrow \frac{k}{k+1}\bar{b}_k + \frac{1}{k+1} b_{k+1}
\end{aligned}
$$

```
# gradient
dw , db = self._SGD(X, y, W, b, batch_idx)

# update (weight averaging)
W = np.subtract(W, np.multiply(self.eta, dw))
b = np.subtract(b, np.multiply(self.eta, db))
W_ = np.add(np.multiply((it/(it+1)), W_), np.multiply((it/(it+1)), W))
b_ = np.add(np.multiply((it/(it+1)), b_), np.multiply((it/(it+1)), b))
```

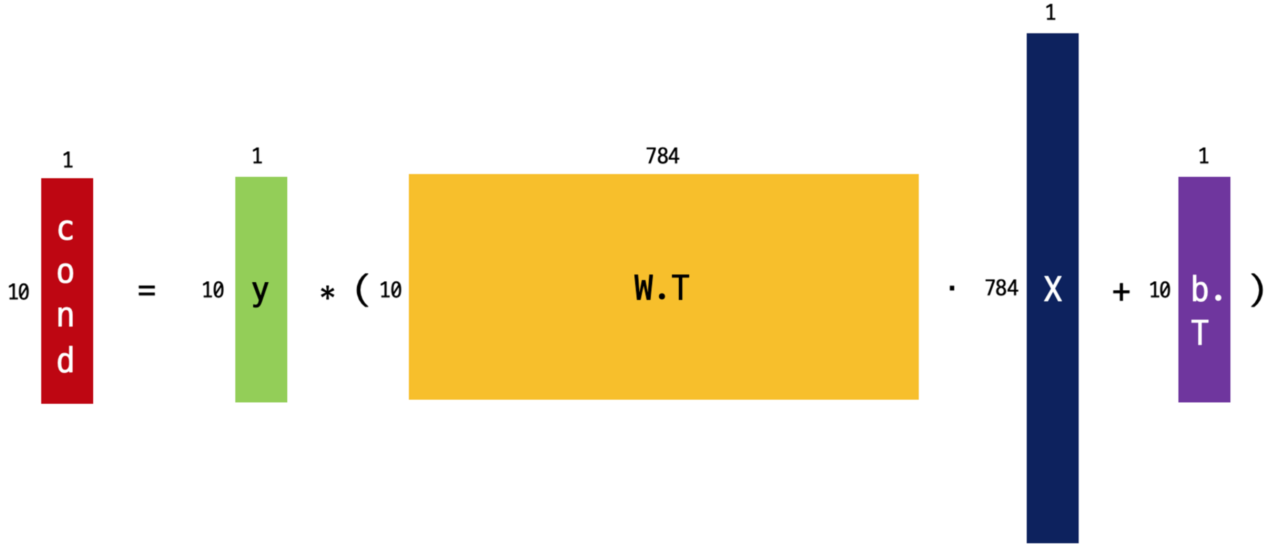
Ⅲ. SGD-SVM

SGDSVM - _SGD() Compute Gradient

```
for idx in batch_idx:
    Xr = X[idx,:].reshape(self.n_features, 1)
    yr = (self.one_hot(y[idx])).reshape(self.n_classes, 1)

    # using chain rule
    z = np.add(np.dot(W.T, Xr), b.T)

    conditions = np.multiply(yr, z)
    conditions[conditions <= 1] = 1 # misclassified
    conditions[conditions > 1] = 0

    v = np.dot(Xr, np.multiply(-yr.T, conditions.T))
    dW = np.add(dW, v)
    db = np.add(db, np.multiply(-yr.T, conditions.T))

dW /= self.batch_size
dW = np.add(dW, np.dot((1 / self.C), W)) # margin
db /= self.batch_size
```

$$\nabla_w \tilde{F}(w_k, b_k) = \frac{1}{|B_k|} \sum_{r \in B_k} \begin{cases} -y_r x_r & \text{if } y_i \langle w_k, x_r \rangle + b_k \leq 1 \\ 0 & \text{o.w.} \end{cases} + \lambda w_k$$

$$\nabla_b \tilde{F}(w_k, b_k) = \frac{1}{|B_k|} \sum_{r \in B_k} \begin{cases} -y_r & \text{if } y_i \langle w_k, x_r \rangle + b_k \leq 1 \\ 0 & \text{o.w.} \end{cases}$$



Ⅲ. SGD-SVM

SGDSVM - fit() Keep the Best

```
# keep the best weight and bias
if self._check_score(X, y, self.best_W, self.best_b) < self._check_score(X, y, W_, b_):
    self.best_W = W_[:]
    self.best_b = b_[:]
```

Ⅲ. SGD-SVM

SGDSVM - predict()

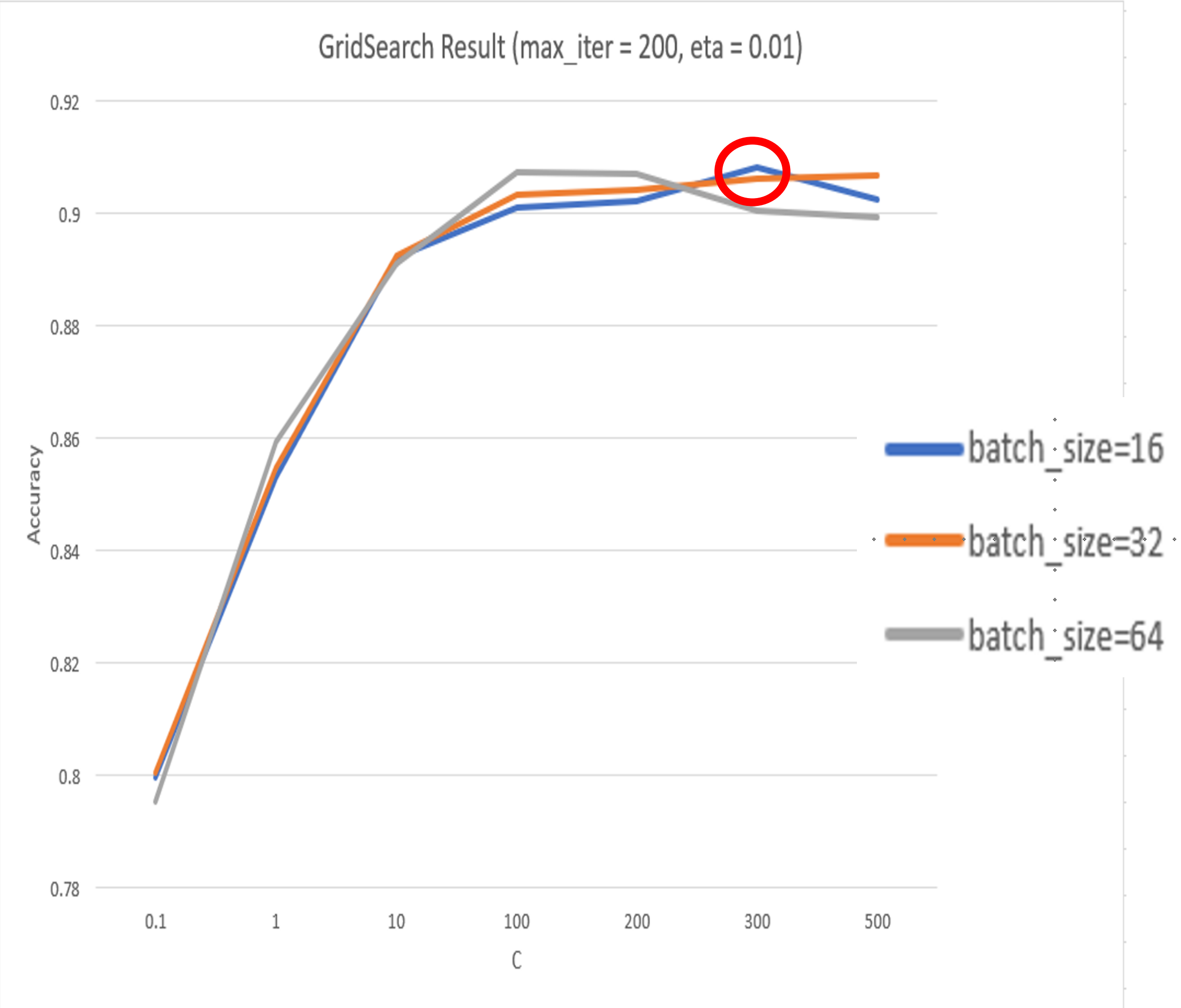
```
def predict(self, X):
    try:
        z = np.dot(X, self.best_W) + self.best_b
        y_pred = np.argmax(z, axis=1)

    except AttributeError:
        raise RuntimeError("You must train classifier before predicting data!")

    return y_pred
```

IV. Hyperparameter Tuning

```
param_grid = [{

    ‘C ’ : [0.1, 1, 10, 100, 200, 300, 500, 1000]
    ‘eta’ : [0.1, 0.01, 0.001]
    ‘max_iter’ : [200]
    ‘batch_size’ : [16, 32, 64, 128]

}]
```

IV. Hyperparameter Tuning

eta	batch_size	max_iter	C	Accuracy
0.01	**16**	**200**	**300**	**0.908208333**
0.01	64	200	100	0.907333333
0.01	64	200	200	0.90725
0.01	128	200	1000	0.906875
0.01	32	200	500	0.906791667
0.01	32	200	300	0.906375
0.01	32	200	200	0.904375
0.01	32	200	100	0.903333333
0.01	16	200	500	0.902416667
0.01	16	200	200	0.902166667
0.01	64	200	1000	0.901875
0.01	16	200	100	0.901
0.01	128	200	100	0.9008



V. Conclusion

SGDSVM(C=300.0, eta=0.01, max_iter=200, batch_size=16)

↓

Accuracy : 0.90533

Q & A

Thanks!