MACHINE LEARNING APPLICATION

01418362 Introduction to Machine Learning

6410401060 Thanaphat Chueatoluang

INTRODUCTION

- **Codeforces** is a competitive programming platform that allows everyone to participate in contests.
- Each user's programming skills are rated by **Ranking**, through their performance in past contests they have participated in.
- **The task is to classify the rank of each user.**

DATASET

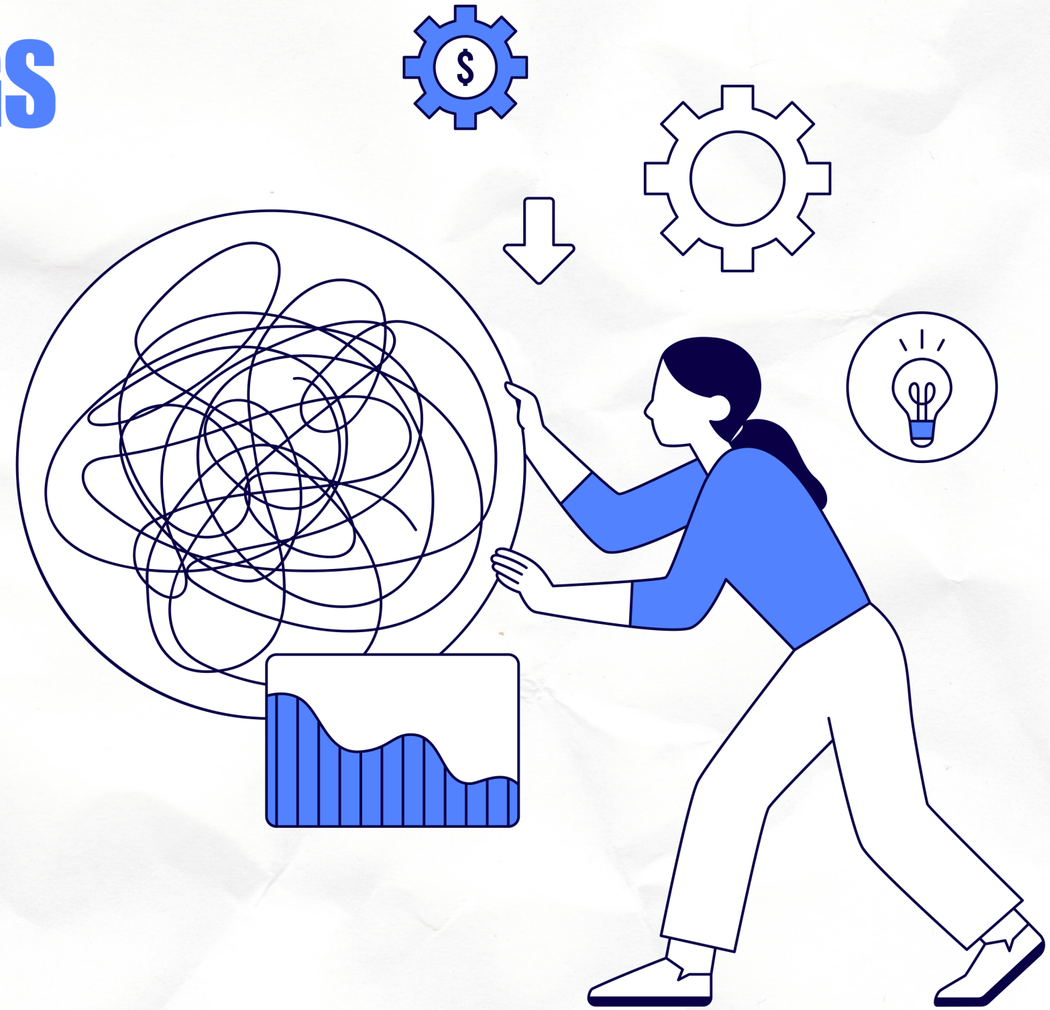
▶▶ KAGGLE: CODEFORCES RATINGS

Contains 1039 user-specific data, along with their rank types and contest performance.

Features:

- userid: Unique identifier for Codeforces users.
- contest1 - contest10: Performance ratings of users in ten different Codeforces contests.
- rank-type: The user's rank type on Codeforces, indicating their skill level.
    - Master: 628
    - Candidate Master: 411

The dataset contains 511 missing values.



DATASET

▶▶ EXAMPLE

	userid	rank-type	contest1	contest2	contest3	contest4	contest5	contest6	contest7	contest8	contest9	contest10
0	3143927301	Candidate Master	2078	2055	2115	2047.0	2024.0	2010.0	1953.0	1936.0	2042.0	2045.0
1	1876577621	Master	2194	2114	2152	2179.0	2211.0	2154.0	2170.0	2141.0	2157.0	2209.0
2	6397741793	Master	2120	2206	2147	2234.0	2294.0	2090.0	2089.0	2072.0	2085.0	2114.0
3	3090123616	Master	2224	2222	2166	2116.0	2029.0	2113.0	2104.0	2096.0	2115.0	2163.0
4	9564162806	Master	2128	2120	2072	2018.0	1963.0	2039.0	1932.0	1963.0	1960.0	1886.0
...	...	...	...	...	...	...	...	...	...	...	...	...
1034	9480350921	Candidate Master	2000	1951	1900	1927.0	1820.0	1787.0	1721.0	1686.0	1681.0	1592.0
1035	7473555183	Candidate Master	1977	1910	1974	1823.0	1796.0	1790.0	1829.0	1717.0	1676.0	1716.0
1036	7812437094	Candidate Master	2066	2098	1888	1788.0	1689.0	1386.0	1111.0	679.0	NaN	NaN
1037	7119479728	Master	2157	2118	1818	1684.0	1387.0	787.0	NaN	NaN	NaN	NaN
1038	1504914733	Master	2243	2191	2052	1936.0	1622.0	1297.0	693.0	NaN	NaN	NaN

PREPROCESSING

- Mean Impute
- Normalization
- Train/Test Split

PREPROCESSING

MEAN IMPUTE

Replacing missing values with the mean of the non-missing values in the same column.

Code Example

```
#Impute missing value by mean
imputer = SimpleImputer(strategy='mean')
impute = ['contest'+str(i) for i in range(1,11)]
df[impute] = imputer.fit_transform(df[impute]).astype(int)
```

PREPROCESSING

NORMALIZATION

Rescale each feature values to normal distribution

$$z_{i,feature} = \frac{x_{i,feature} - mean_{feature}}{SD_{feature}}$$

Code Example

```
#Normalize all feature to z-value
scaler = StandardScaler()
df[impute] = scaler.fit_transform(df[impute])
```

PREPROCESSING

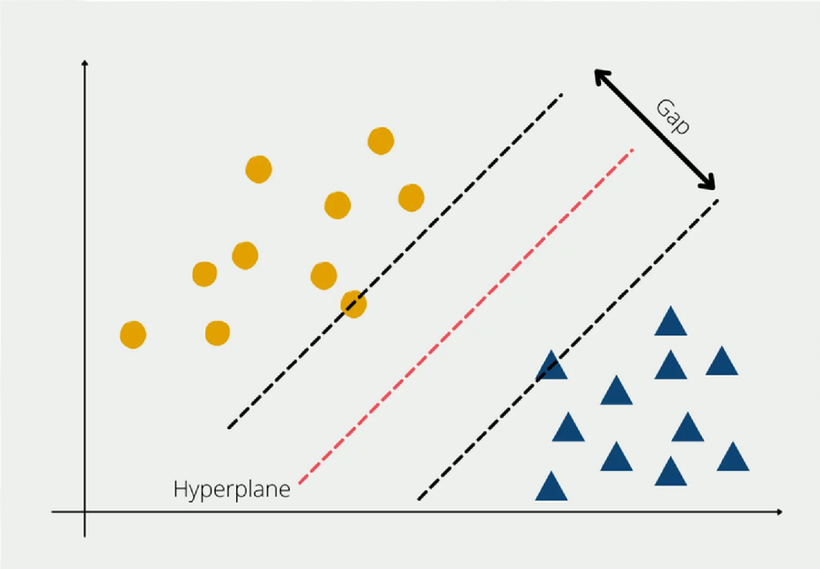
TRAIN/TEST SPLIT

Code Example

```python
def train_test_split(x,y,test_size):
    indices = np.arange(len(x))
    np.random.shuffle(indices)
    x_shuffled = x[indices]
    y_shuffled = y[indices]
    pos = int(y.shape[0]*(1-test_size)) #number of train size
    return x_shuffled[:pos],x_shuffled[pos:],y_shuffled[:pos],y_shuffled[pos:]
```

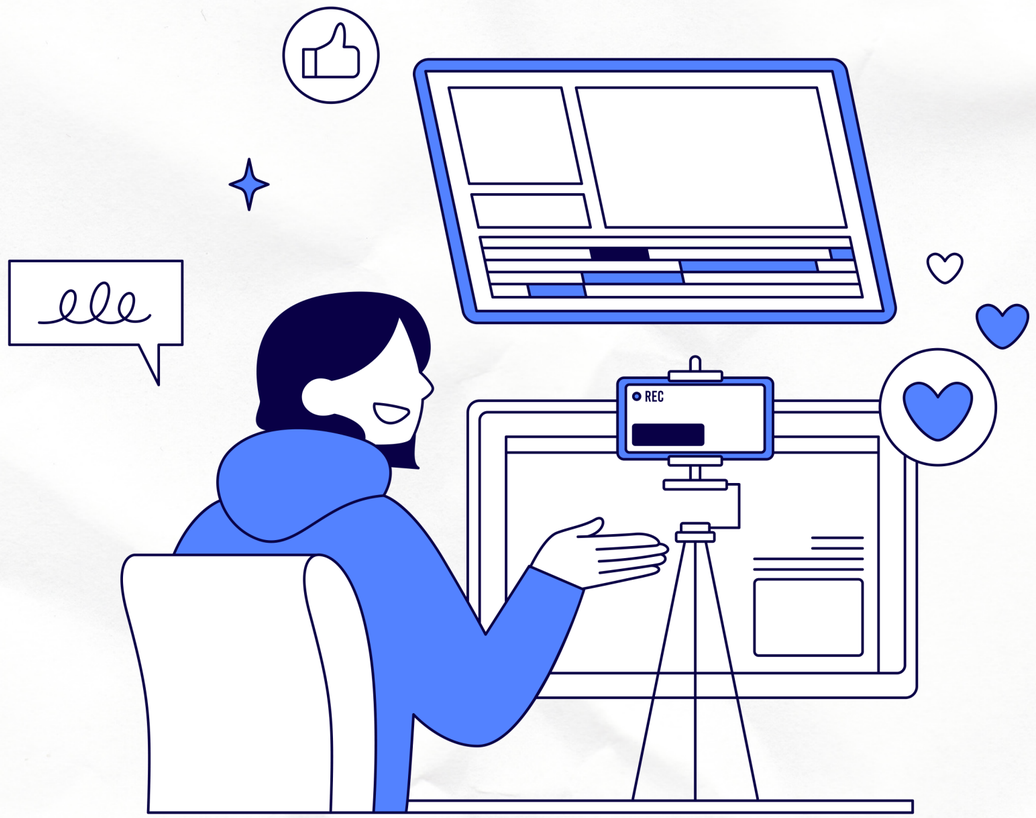
MODEL SELECTION
SUPPORT VECTOR MACHINE

- **Binary Classification**
- **High-Dimensional Dataset**



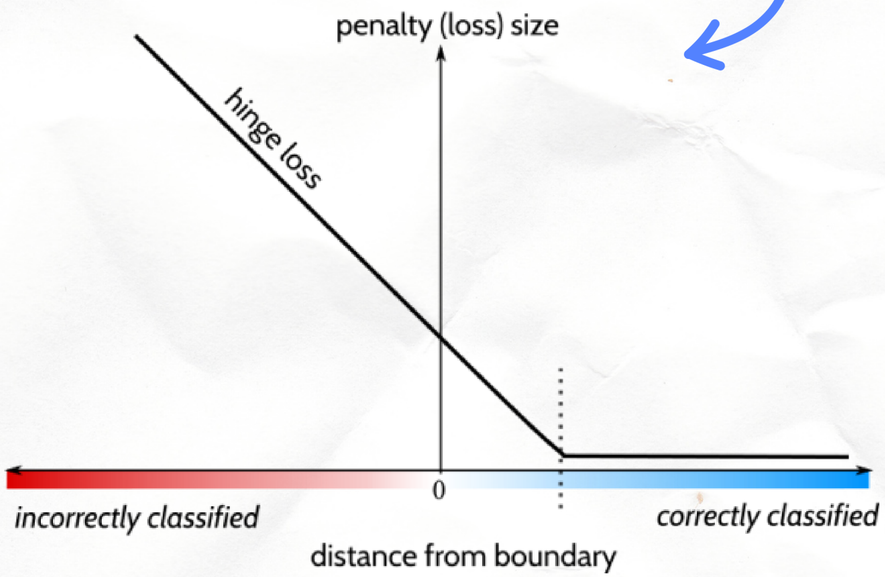
SUPPORT VECTOR MACHINE

RECAP (SOFT CONSTRAINT)

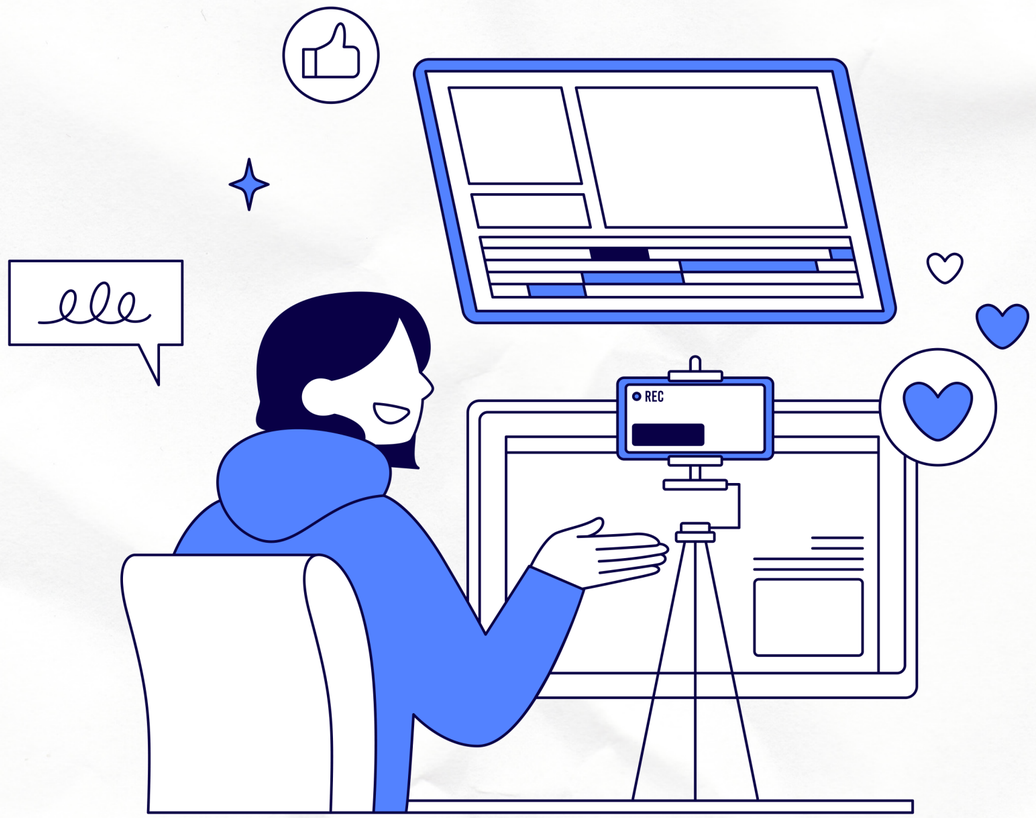


Soft Constraint SVM try to minimize this

$$\min_{w} \quad w^T w \quad + \quad C * \sum_{i} [\max(0, 1 - y_i (w^T x_i))]$$



DERIVING



$$\min_{w} \quad w^T w + C * \text{sum}_i [\max(0, 1 - y_i(w^T x_i))]$$

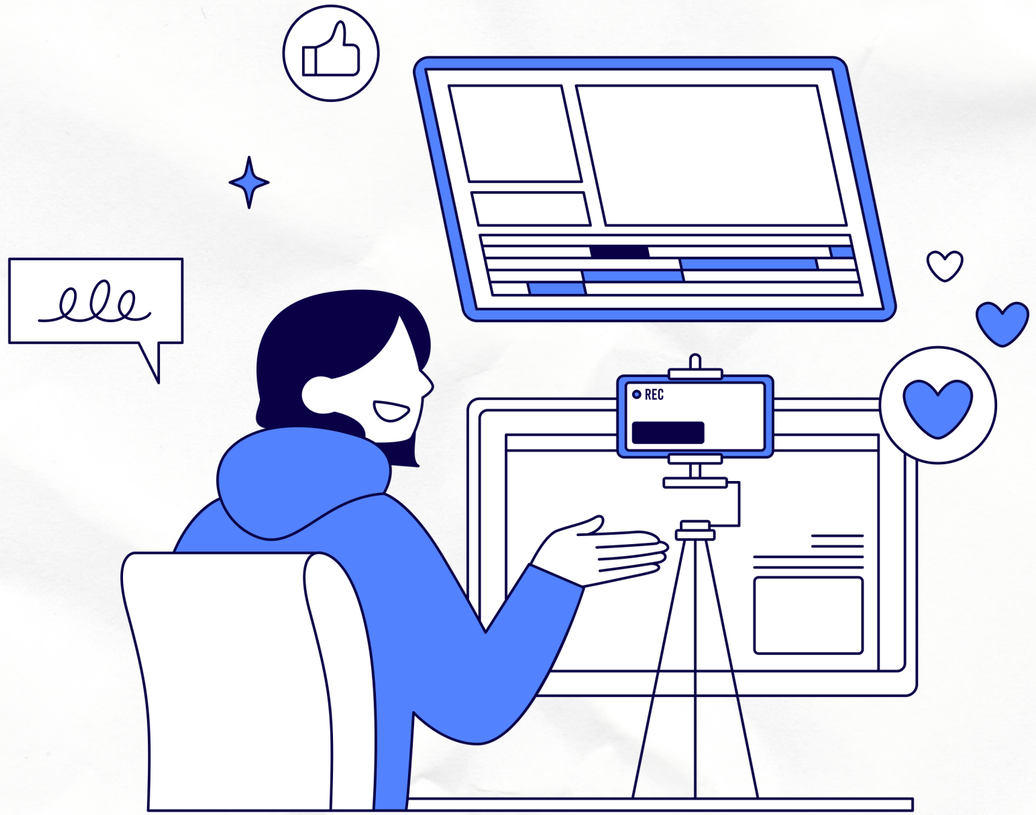
$$\min_{w} \quad \lambda * w^T w + \text{sum}_i [\max(0, 1 - y_i(w^T x_i))]$$

taking $\lambda = 0$;

$$\min_{w} \quad \text{sum}_i [\max(0, 1 - y_i(w^T x_i))]$$

(minimizing loss)

DERIVING



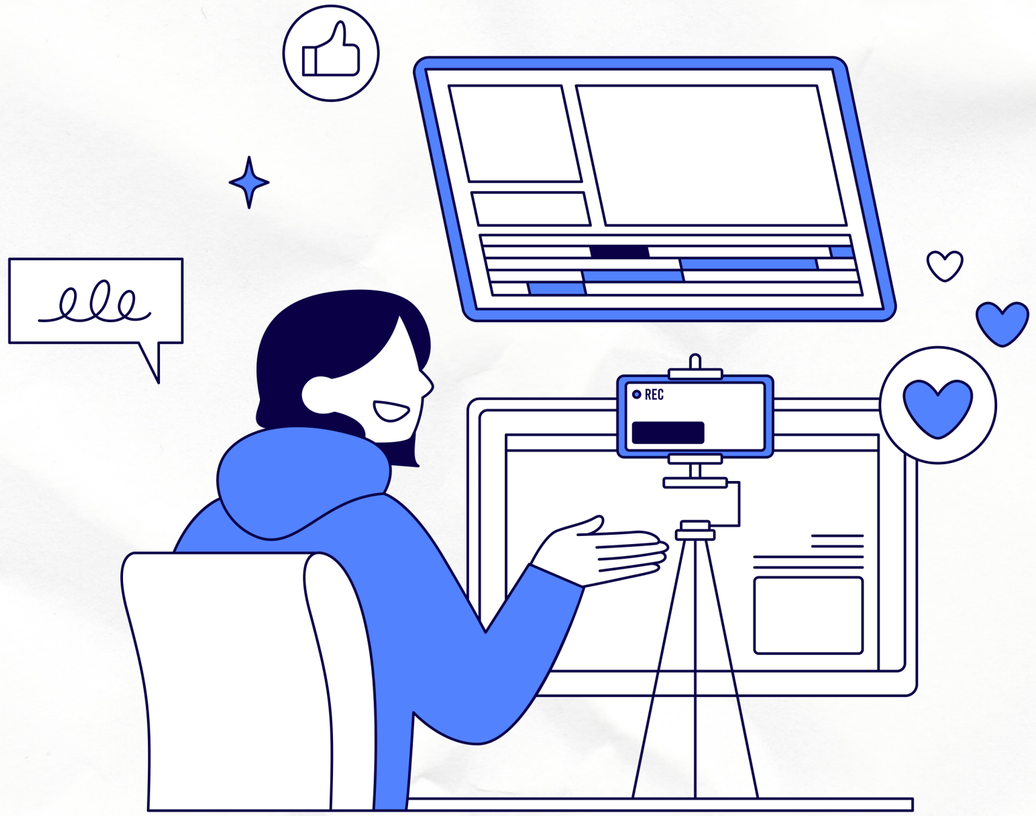
$$\min_{w} \quad \sum_{i} [\max(0, 1 - y_i(w^T x_i))]$$

consider $f = \sum_{i} [\max(0, 1 - y_i(w^T x_i))]$

$$\frac{df}{dw_j} = \begin{cases} 0 & \text{if} \quad y_i(w^T x_i) >= 1 \\ -x\ y & \text{if} \quad y_i(w^T x_i) < 1 \end{cases}$$

f is continuous, convex, and differentiable
so Gradient Descent is useable

GRADIENT DESCENT
HILL CLIMBING ALGORITHM



Initialize:

$w = \{0,\ldots,0\}$

Maintain:

while $||w_{t+1} - w_t||_2 >= \delta$:

$w_{t+1} = w_t - \alpha * \text{gradient}(\text{loss}(w_t))$

MODEL CODE EXAMPLE

```python
class SVM:
    def __init__(self, d, a):
        self.delta = d
        self.alpha = a

    def gradient(self, x, y):
        g = np.zeros(self.w.shape[0])
        for xi,yi in zip(x,y):
            if np.sum(np.dot(self.w,xi))*yi < 1:
                g -= xi*yi
        return g

    def l2norm(self, X):
        return np.sqrt(np.sum(np.dot(X,X)))

    def fit(self, X, y):
        self.X = X[:]
        self.y = y[:]
        self.train()
```

```python
def train(self):
    self.class_value = {}
    for i,c in enumerate(np.unique(self.y)):
        self.class_value[2*i-1] = c
        self.y[self.y == c] = 2*i-1
    self.w = np.zeros(self.X.shape[1])

    while True:
        grad = self.gradient(self.X, self.y)
        w_new = self.w - self.alpha * grad
        if self.l2norm(w_new-self.w) < self.delta:
            break
        self.w = w_new.copy()

def predict(self, X):
    y_pred = np.sign(np.dot(X, self.w)+1e-9).astype(int)
    return [self.class_value[pred] for pred in y_pred]

def accuracy_score(self, X, y):
    y_pred = self.predict(X)
    return np.sum(y_pred == y)/y.shape[0]
```

RESULT

for this dataset, accuracy is around 80%-90%

```
classifier = SVM(1e-3,1e-5)
classifier.fit(X_train,y_train)
```

✓ 3.7s

```
print('Accuracy:',classifier.accuracy_score(X_test, y_test))
```

✓ 0.0s

```
Accuracy: 0.8942307692307693
```

THANK YOU