Date: / /

Random Forest Algorithm

- For each tree $t = 1$ to T
- Draw a bootstrap sample D_t of size n from training data D (sampling with replacement)
- Train a decision tree h_t on D_t:
  At each node:
  Randomly select m features from the full feature set.
  Choose the best feature & split point among m features using Impurity, Entropy or MSE (for regression)
- Split the node into child nodes

2) At Aggregate the predictions:
For classification: Use majority voting over all trees:

$$\hat{y} = \text{mode}(h_1(x), h_2(x), \ldots, h_T(x))$$

For regression: Use mean prediction:

$$\hat{y} = \frac{1}{T} \sum_{t=1}^{T} h_t(x)$$

Code:

Class simple Random Forest:

```
def init(self, n_trees=10, max_features=
    sqrt):
    self.n_trees = n_trees
    self.max_features = max_features
    self.trees = []
```

papergrid
Date / /

```
def _bootstrap_sample(self, X, y):
    n_samples = X.shape[0]
    indices = np.random.choice(n_samples,
    size = n_samples, replace = True)
    return X[indices], y[indices]
def _get_max_features(self, n_features):
    if self.max_features == 'sqrt':
        return int(np.sqrt(n_features))
    elif isinstance(self.max_features, int):
        return self.max_features
def fit(self, X, y):
    self.trees = []
    n_features = X.shape[1]
    max_feats = self._get_max_features
```

```
def predict(self, X)
    tree_preds = np.array([tree.predict(X)
    for tree in self.trees])
    return np.apply_along_axis(self.majority_vote,
    axis=0, arr = tree_preds)
```

K means Algorithm

1) Initialize centroids:
Randomly choose K data points from X as initial cluster centroids:

$$\mu_1, \mu_2, \ldots \mu_K$$

2) Repeat until convergence:
a. Assign each data point to the nearest centroid:
For each point x_i, find the closest centroid μ_j based on distance

Date: / /

Cluster $(x_i) = \arg\min_j \|x_i - \mu_j\|^2$

b) Update centroids
Recalculate centroid of each cluster by computing mean of all points

$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

3) Check for convergence:
* If cluster assignments don't change
* Centroids don't move significantly, then stop

Code

```
import matplotlib.pyplot as plt
X, _ = make_blobs(n_samples=300,
centers=3, random_state=42)
model = KMeans(n_clusters=3)
model.fit(X)
plt.scatter(X[:,0], X[:,1],
c=model.labels_, cmap='viridis')
plt.show()
```

Ada Boosting ensemble Algorithm

1) Initialize sample weights
$w_i^{(1)} = \frac{1}{n}$ for $i = 1, 2, \ldots n$

2) Repeat for $t = 1$ to T (Number of weak learners)

a) Train a weak learner $h_t(x)$
Train the model using current weights $w_i^{(t)}$

papergrid
Date: / /

b Compute the weighted error of weak learner

$$\varepsilon_t = \sum_{i=1}^{n} w_i(t) \cdot \mathbb{I}(h_t(x_i) \neq y_i)$$

$\mathbb{I}$ (condition) = 1 if condition is true else 0

c compute the learner's weight

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

d Update sample weights

$$w_i^{(t+1)} = w_i^{(t)} \cdot e^{-\alpha_t \cdot y_i \cdot h_t(x_i)}$$

Misclassified = higher weights
classified = lower weights

e Normalize weights sum to 1:

$$w_i^{(t+1)} \leftarrow \frac{w_i^{(t+1)}}{\sum_{j=1}^{n} w_j^{(t+1)}}$$

3) Final strong classifier (output)

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \cdot h_t(x)\right)$$

Code

```
X, y = make_classification(n_samples=500,
    n_features=10, random_state=42)
X_train, X_test, y_train, y_test = train_test_
    split(X, y, test_size=0.3, random_state
    =42)
weak_learner = DecisionTreeClassifier
    (max_depth=1)
model = AdaBoostClassifier(base_estimator
    =weak_learner, n_estimators=50)
model.fit(X_train, y_train)
```

Date: / /

Principle component Model Algorithm

1) Standardize the data
PCA is affected by scale of features
Therefore data should be standardized

$$x_j = \frac{x_j - \mu_j}{\sigma_j}$$

2) Compute the covariance matrix

$$\Sigma = \frac{1}{n-1} X^T X$$

3) Compute the eigen values and eigen vectors of covariance matrix

$$\Sigma v = \lambda v$$

4) Sort the eigen values & vectors in descending order

5) Select the top k eigen vectors

6) Project the data onto new space

7) Output the transformed data

Code

```
data = load_iris()
X = data.data
pca = PCA(n_components = 2)
X_pca = pca.fit_transform(X)
```

papergrid
Date: / /

```
X_pca = pca.fit_transform(x)
plt.scatter(X_pca[:,0], X_pca[:,1],
  c = data.target)
print(f'Explained variance by each
  component : {}')
```

21/4/25