Hyperparameter Tunning

Machine Learning for Engineering Applications

Fall 2023

Class Imbalance

- This a major problem with all dataset, unless you have the perfect one

- 99.9999% of all datasets suffer of class imbalance

- Class(es) with the majority of the distribution can *influence* the training and biasing of the model.

- There are some techniques that help balance, and the goal of the designer is to find a balance

- One technique: Hand-balance the classes

- Example:
  - Cancer data:
    - 357 samples of the benign class (class 0)
    - 212 samples of the malignant class (class 1)

```
X_imb = np.vstack((X[y == 0], X[y == 1][:40]))
y_imb = np.hstack((y[y == 0], y[y == 1][:40]))

y_pred = np.zeros(y_imb.shape[0])

np.mean(y_pred == y_imb) * 100
89.92443324937026 7
```

- ML techniques use the *cost* or *reward* functions as a sum for the training samples during training

- This will make the model *biased* to the over-represented class(es)

- One can assign *penalties* or *regulations* to force the cost functions to restrict a major reward towards specific weights

- The only way to know: experimentation with different strategies

- Another technique: Resampling

- Example:
  - 357 samples of the benign class (class 0)
  - 212 samples of the malignant class (class 1)

- Resampling takes the minority class and repeatedly picks new samples from it until it research the majority number

- In this case: whenever it reaches 357 of class 1

Another technique: Resampling

```
from sklearn.utils import resample

X_imb = np.vstack((X[y == 0], X[y == 1][:40]))
y_imb = np.hstack((y[y == 0], y[y == 1][:40]))
print('Number of class 1 samples before:’, X_imb[y_imb ==
                                            ... 1].shape[0])
Number of class 1 samples before: 40

X_upsampled, y_upsampled = resample(X_imb[y_imb == 1],
     ... y_imb[y_imb == 1], replace=True,
     ... n_samples=X_imb[y_imb == 0].shape[0],random_state=123)

print('Number of class 1 samples after:’, X_upsampled.shape[0])
Number of class 1 samples after: 357
```

Continuation….

```
X_upsampled, y_upsampled = resample(X_imb[y_imb == 1],
      ... y_imb[y_imb == 1],
      ... replace=True,
      ... n_samples=X_imb[y_imb == 0].shape[0],
      ... random_state=123)

#stack the original class 0 samples with the upsampled class 1 subset
X_bal = np.vstack((X[y == 0], X_upsampled))
y_bal = np.hstack((y[y == 0], y_upsampled))

#a majority vote prediction rule would only achieve 50 percent accuracy
y_pred = np.zeros(y_bal.shape[0])
np.mean(y_pred == y_bal) * 100
```

Grid Search

- Fine tuning your parameters with Grid Search
- Hand tuning can help but it takes a long time!
- Best thing: Let me machine do the work for you.

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
pipe_svc = make_pipeline(StandardScaler(), SVC(random_state=1))
param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
param_grid = [{'svc__C': param_range,
      ... 'svc__kernel': ['linear']},
      ... {'svc__C': param_range, 'svc__gamma': param_range,
      ... 'svc__kernel': ['rbf’]}]
gs = GridSearchCV(estimator=pipe_svc,
      ... param_grid=param_grid,
      ... scoring='accuracy’,
      ... cv=10,
      ... n_jobs=-1)
gs = gs.fit(X_train, y_train)
print(gs.best_score_)
0.984615384615
print(gs.best_params_)
{'svc__C': 100.0, 'svc__gamma': 0.001, 'svc__kernel': 'rbf'}
```