Exercise Set 5b

Hina Arora

(5b)
Cross-Validation

https://scikit-learn.org/stable/modules/cross_validation.html

[Note:
Cross-Validation is typically done in conjunction with Grid-Search, as we'll see in (5d)]

Hyper-Parameter Tuning

“Hyper-parameters are parameters that are not directly learnt within estimators. In scikit-learn they are passed as arguments to the constructor of the estimator classes.

When evaluating hyperparameters for estimators, there is a risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally. This way, knowledge about the test set can “leak” into the model and evaluation metrics no longer report on generalization performance.

To solve this problem, yet another part of the dataset can be held out as a so-called “validation set”: training proceeds on the training set, after which evaluation is done on the validation set, and when the experiment seems to be successful, final evaluation can be done on the test set.

However, by partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets.”

Cross-Validation

“A solution to this problem is a procedure called Cross-Validation. A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV.

In the basic approach, called k-fold CV, the training set is split into k smaller sets. The following procedure is followed for each of the k “folds”:

- A model is trained using k-1 of the folds as training data;
- The resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).

The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop. This approach can be computationally expensive, but does not waste too much data (as is the case when fixing an arbitrary validation set), which is a major advantage in problems such as inverse inference where the number of samples is very small.”

Copy

```
df = pd.read_csv('data/kaggleTitanic/train.csv’)

df['Deck'] = df['Cabin'].apply(lambda x: x[0] if pd.notna(x) else np.nan)

X = df.drop(['Survived'], axis=1)
y = df['Survived’]

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, random_state=1)

Xtrain = Xtrain.copy()
Xtest = Xtest.copy()
ytrain = ytrain.copy()
ytest = ytest.copy()
```

Copy

```
numeric_features = ['Age']
numeric_transformer = Pipeline(steps=[
    ('si', SimpleImputer(missing_values=np.nan, strategy='median'))])

categorical_features = ['Pclass', 'Sex', 'Deck']
categorical_transformer = Pipeline(steps=[
    ('si', SimpleImputer(missing_values=np.nan, strategy='constant', fill_value='X')),
    ('ohe', OneHotEncoder(sparse=False, dtype=int, handle_unknown='ignore'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)],
    remainder='drop’)

clf = Pipeline(steps=[('pp', preprocessor),
                      ('lr', LogisticRegression(solver='liblinear'))])
```

Copy

```
# Use 5-fold cross-validation:
#      train, validate each time and get the mean scores

from sklearn.model_selection import cross_validate

scores = cross_validate(
                clf,
                Xtrain, ytrain,
                cv=5,
                return_train_score=False)
scores['test_score'].mean()
```