CPSC 330

Lecture 7: Linear Models

2026-01-31

In linear regression models we learn the coefficients and the intercept of the line of best fit.

- In SciKit learn, we use Ridge. It is a linear regression model with the hyper parameter α to regulate learning.
  - Large alpha $\rightarrow$ likely to underfit (more bias for high values of α)

Linear models include:

- Linear regression
- Logistic regression
  - Applies a threshold on raw output data to determine wehter the class is positive or negative
- Linear SVM

Logistic Regression

Prediction is based on weighted sum of input features.
Encode what it means to be positve or negative
Some features pull toward positive or negative sentiment, if result < 0, then classified negative and determined to be whatever we encoded into negative.
Decision boundary is a hyperplane dividing the feature space in half

- C is the main hyperparameter
  - small C $\rightarrow$ underfitting
  - bigger C $\rightarrow$ overfitting
  - To tune for the best C, we test C's of different orders of magnitude $\{10^0, 10^1, 10^2, ...\}$

Prediciting probabilities

```
1 lr.predict_proba([example])
```

returns the probability that a data point belongs to each class

- For logistical regression we check the sign of the raw model output (these are hard preditions)
  - To convert raw model output into probabilities, instead of taking the sign we apply the sigmoid function
- When we plot the sigmoid function, if the decision boundary is very clear cut, it has a steep increase at the decision boundary
- If the decision boundary is a bit weaker, and there is more overlap between output classes, the sigmoid function has a more gradual increase

Hyperparameter Optimization

```
GridSearchCV
```

- We need an instantiated model or pipeline
- a parameter grid: a user specifies a set of values for each hyperparameter

```
from sklearn.model_selection import GridSearchCV

pipe_svm = make_pipeline(preprocessor, SVC())

param_grid = {
    "columntransformer__countvectorizer__max_features": [100, 200, 400, 800, 1000, 2000],
    "svc__gamma": [0.001, 0.01, 0.1, 1.0, 10, 100],
    "svc__C": [0.001, 0.01, 0.1, 1.0, 10, 100],
}

# Create a grid search object
gs = GridSearchCV(pipe_svm,
                  param_grid = param_grid,
                  n_jobs=-1,
                  return_train_score=True
                 )
gs.best_score_
gs.best_params_
results = pd.DataFrame(gs.cv_results_)
results.T
gs.score(X_test, y_test) # gs also trains the model on the best scores
```

- SVC is quite sensitive to hyperparameter tuning

```
param_grid3 = {"svc__gamma": np.logspace(-3, 2, 6), "svc__C": np.linspace(1, 10, 6)}

display_heatmap(param_grid3, pipe_svm, X_train, y_train)
```

Randomized hyperparameter search

- Samples configurations at random until cetain budget is exhausted (e.g. time)

```
from sklearn.model_selection import RandomizedSearchCV

param_grid = {
```

```
 4     "columntransformer__countvectorizer__max_features": [100, 200, 400, 800,
   1000, 2000],
 5     "svc__gamma": [0.001, 0.01, 0.1, 1.0, 10, 100],
 6     "svc__C": [0.001, 0.01, 0.1, 1.0, 10, 100]
 7 }
 8
 9 print("Grid size: %d" % (np.prod(list(map(len, param_grid.values())))))
10 param_grid
```

- We can pass in probability distributions to random seach to draw our data points from
- For C and Gamma, we want to use logarithmic and not linear hyperparameter testing space

Advantages of Random Search over Grid Search

- Faster
- Adding parameters does not influence the performance, does not affect the efficiency
- Works better when some parameters are more important than others

In class questions:

- Suppose you have 10 hyperparameters, each with 4 possible values. If you run GridSearchCV with this parameter grid, how many cross-validation experiments will be carried out?
  - 4^{10}
- Suppose you have 10 hyperparameters and each takes 4 values. If you run RandomizedSearchCV with this parameter grid with n_iter=20, how many cross-validation experiments will be carried out?
  - Total experiments = n_iter $\times$ cv
  - We specify the number of cross validations

Overfitting of the validation error

- If our dataset is small and if your validation set is hit too many times, we suffer from optimization bias or overfitting the validation set
- During training, we could search over tons of different decision trees.
- So we can get "lucky" and find a tree with low training error by chance.
- This is the reason behind cross-validation: we do not want to trust a single fit.

Optimization bias grows with the number of things that we try (we select the best performance), BUT, the optimization shrinks quickly with the number of samples. (but its still non-zero and growing if you over-use your validation error)