Summary Session WEEK-7 (Part-A)

By - Jimmi

Contents:

- Softmax Regression
- Nearest Neighbour Classifier
- Large scale machine learning

Softmax regression:

- Softmax regression can be used to perform multiclass classification.

- To use Softmax activation set multiclass parameter to ‘multinomial’ .

```
pipe = Pipeline(steps=[('scaler', StandardScaler()),
                       ('logreg', LogisticRegression(multi_class='multinomial',
                                                     solver='sag'))])
pipe.fit(x_train, y_train)
```

	Solvers				
Penalties	**'liblinear'**	**'lbfgs'**	**'newton-cg'**	**'sag'**	**'saga'**
Multinomial + L2 penalty	no	yes	yes	yes	yes
OVR + L2 penalty	yes	yes	yes	yes	yes
Multinomial + L1 penalty	no	no	no	no	yes
OVR + L1 penalty	yes	no	no	no	yes
Elastic-Net	no	no	no	no	yes
No penalty ('none')	no	yes	yes	yes	yes
Behaviors					
Penalize the intercept (bad)	yes	no	no	no	no
Faster for large datasets	no	no	no	yes	yes
Robust to unscaled datasets	yes	yes	yes	no	no

Nearest neighbor classifier

- It is a type of instance-based learning or non-generalizing learning
  - does not attempt to construct a model
  - simply stores instances of the training data

- Classification is computed from a simple majority vote of the nearest neighbors of each point.

- Two different implementations of nearest neighbors classifiers are available.

1. KNeighborsClassifier
2. RadiusNeighborsClassifier

How are KNeighborsClassifier and RadiusNeighborsClassifier different?

KNeighborsClassifier	RadiusNeighborsClassifier
• learning based on the k nearest neighbors	• learning based on the number of neighbors within a fixed radius *r* of each training point
• most commonly used technique	• used in cases where the data is not uniformly sampled
• choice of the value *k* is highly data-dependent	• fixed value of *r* is specified, such that points in sparser neighborhoods use fewer nearest neighbors for the classification

How do you apply KNeighborsClassifier?

Step 1: Instantiate a KNeighborsClassifer estimator without passing any arguments to it to create a classifer object.

```
1 from sklearn.neighbors import KNeighborsClassifier
2 kneighbor_classifier = KNeighborsClassifier()
```

Step 2: Call fit method on KNeighbors classifier object with training feature matrix and label vector as arguments.

```
1 # Model training with feature matrix X_train and
2 # label vector or matrix y_train
3 kneighbor_classifier.fit(X_train, y_train)
```

How do you specify the number of nearest neighbors in KNeighborsClassifier?

- Specify the number of nearest neighbors *K* from the training dataset using `n_neighbors` parameter.
  - value should be int.

```
1 kneighbor_classifier = KNeighborsClassifier(n_neighbors = 3
```

What is the default value of *K*?

`n_neighbors = 5`

How do you assign weights to neighborhood in KNeighborsClassifier?

- It is better to weight the neighbors such that nearer neighbors contribute more to the fit.

`weights`

- 'uniform' : All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance.
  - closer neighbors of a query point will have a greater influence than neighbors which are further away.

Default:

```
1 kneighbor_classifier = KNeighborsClassifier(weights= 'uniform')
```

Can we define our own weight values for KNeighborsClassifier?

- Yes, it is possible if you have an array of distances.
- `weights` parameter also accepts a user-defined function which takes an array of distances as input, and returns an array of the same shape containing the weights.

Example:

```
1 def user_weights(weights_array):
2     return weights_array
3
4 kneighbor_classifier = KNeighborsClassifier(weights=user_weights)
```

How do you apply RadiusNeighborsClassifier?

Step 1: Instantiate a RadiusNeighborsClassifer estimator without passing any arguments to it to create a classifer object.

```
1 from sklearn.neighbors import RadiusNeighborsClassifier
2 radius_classifier = RadiusNeighborsClassifier()
```

Step 2: Call fit method on RadiusNeighbors classifier object with training feature matrix and label vector as arguments.

```
1 # Model training with feature matrix X_train and
2 # label vector or matrix y_train
3 radius_classifier.fit(X_train, y_train)
```

How do you specify the number of neighbors in RadiusNeighborsClassifier?

- The number of neighbors is specified within a fixed radius *r* of each training point using `radius` parameter.
- *r* is a float value.

```
1 radius_classifier = RadiusNeighborsClassifier(radius=1.0)
```

What is the default value of *r* ?

`r = 1.0`

Large-scale machine learning:

- Scikit-learn handles large data through Partial_fit() method instead of using the usual fit() method.
- The idea is to process data in **batches** and **update** the model parameters for each batch.
- This way of learning is referred to as '**Incremental (or out-of-core) learning**'.

The following estimator implement partial_fit method:

- **Classification:**
  - Multinomial NB
  - BernoulliNB
  - SGDClassifier
  - Perceptron
- **Regression:**
  - SGDRegressor
- **Clustering:**
  - MiniBatchKMeans

```
import pandas as pd

chunksize = 1000

iter = 1
for train_df in pd.read_csv("train_data.csv", chunksize=chunksize,
                            iterator=True):
      xtrain_partial = train_df.iloc[:, 0:10]
      ytrain_partial = train_df.iloc[:, 10]
      clf2.partial_fit(xtrain_partial, ytrain_partial)

    print("After iter #", iter)
    print(clf2.coef_)
    print(clf2.intercept_)
    iter = iter + 1
```

Vectorizer

- Vectorizers are used to convert a collection of text documents to a vector representation, thus helping in preprocessing them before applying any model on these text documents.
- 1) CountVectorizer
- 2) HashingVectorizer

Main difference between both is that HashingVectorizer does not store the resulting vocabulary (i.e. the unique tokens).