**SVM TECHNIQUES**

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

Effective in high dimensional spaces.Still effective in cases where number of dimensions is greater than the number of samples.Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation

The support vector machines in scikit-learn support both dense (numpy.ndarray and convertible to that by numpy.asarray) and sparse (any scipy.sparse) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data. For optimal performance, use C-ordered numpy.ndarray (dense) or scipy.sparse.csr\_matrix (sparse) with dtype=float64.

**Linear Regression**

LinearRegression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

**Parameters**

**1.fit\_interceptbool, default=True**

Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations

**2.normalizebool, default=False**

This parameter is ignored when fit\_intercept is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize.

**3. copy\_Xbool, default=True**

If True, X will be copied; else, it may be overwritten.

**4. n\_jobsint, default=None**

The number of jobs to use for the computation. This will only provide speedup for n\_targets > 1 and sufficient large problems. None means 1 unless in a joblib.parallel\_backend context. -1 means using all processors.

**5. positivebool, default=False**

When set to True, forces the coefficients to be positive. This option is only supported for dense arrays.

**ATTRIBUTES**

**1.coef\_array of shape (n\_features, ) or (n\_targets, n\_features)**

Estimated coefficients for the linear regression problem. If multiple targets are passed during the fit (y 2D), this is a 2D array of shape (n\_targets, n\_features), while if only one target is passed, this is a 1D array of length n\_features.

**2.rank\_int**

Rank of matrix X. Only available when X is dense.

**3.singular\_array of shape (min(X, y),)**

Singular values of X. Only available when X is dense.

**4.intercept\_float or array of shape (n\_targets,)**

Independent term in the linear model. Set to 0.0 if fit\_intercept = False.

**KNN TECHNIQUE:**

KNN is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution. In other words, the model structure determined from the dataset. All training data used in the testing phase. This makes training faster and testing phase slower and costlier. Costly testing phase means time and memory. In the worst case, KNN needs more time to scan all data points and scanning all data points will require more memory for storing training data. In KNN, K is the number of nearest neighbors. The number of neighbors is the core deciding factor. K is generally an odd number if the number of classes is 2. When K=1, then the algorithm is known as the nearest neighbor algorithm. This is the simplest case. Suppose P1 is the point, for which label needs to predict. First, you find the one closest point to P1 and then the label of the nearest point assigned to P1. Suppose P1 is the point, for which label needs to predict. First, you find the k closest point to P1 and then classify points by majority vote of its k neighbors. Each object votes for their class and the class with the most votes is taken as the prediction. For finding closest similar points, you find the distance between points using distance measures such as Euclidean distance, Hamming distance, Manhattan distance and Minkowski distance. KNN has the following basic steps:

1. Calculate distance
2. Find closest neighbors
3. Vote for labels

**Multinomial NB;**

implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice). The distribution is parametrized by vectors *θy=(θy1,…,θyn)* for each class *y*, where *n* is the number of features (in text classification, the size of the vocabulary) and *θyi* is the probability *P(xi∣y)* of feature *i* appearing in a sample belonging to class *y*.

The parameters *θy* is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

*θ^yi=Nyi+αNy+αn*

where *Nyi=∑x∈Txi* is the number of times feature *i* appears in a sample of class *y* in the training set *T*, and *Ny=∑i=1nNyi* is the total count of all features for class *y*.

The smoothing priors *α≥0* accounts for features not present in the learning samples and prevents zero probabilities in further computations. Setting *α=1* is called Laplace smoothing, while *α<1* is called Lidstone smoothing.