Introduction to Machine Learning Fall 2019 Problem Set 5
Cole Smith css459

---

1 Kernels

1. This function is a kernel. The encoding Φ can be a vector where each value represents a word with a 1 if the word exists in the document, and 0 if not.

   The kernel $k(x, z)$ is the dot product between two of these encodings, which will return the size of intersection of words contained in both documents.

2. This is a modified polynomial kernel.

$$k_\beta(\vec{x}, \vec{z}) = (1 + \beta\vec{x} \cdot \vec{z})^2 - 1$$

   We can break the dot product $\vec{x} \cdot \vec{z}$ into a summation:

$$\vec{x} \cdot \vec{z} = \sum_i^n x_i z_i$$

   Now we can apply this property to the expanded form, $1 + 2\beta\vec{x} \cdot \vec{z} + (\beta\vec{x} \cdot \vec{z})^2 - 1$. Combining terms:

$$k_\beta(\vec{x}, \vec{z}) = \sum_{i=1}^n \beta^2 x_i^2 z_i^2 + \sum_{i=2}^n \sum_{j=1}^{i-1} (\sqrt{2\beta} x_i x_j)(\sqrt{2\beta} z_i z_j) + \sum_i^n (\sqrt{2\beta} x_i)(\sqrt{2\beta} z_i)$$

   (The two constant 1s cancel)

   This gives the feature map:

$$\Phi(x) =$$

$$(\sqrt{2\beta}x_1,\ \sqrt{2\beta}x_2,\ x_1^2,\ \sqrt{2\beta}x_1x_2,\ x_2^2)$$

   Which is used to map both x and z.

3. Knowing that $k(x,z) = x^T z$ is a kernel, we can work outermost to innermost to prove that the following is also a kernel.

   If we assume that

$$k_1 = 1 + (\frac{x}{\|x\|_2})^T(\frac{z}{\|z\|_2})$$

   is a kernel, then k_1 to the third power satisfies rule iii:

$$k_1^3 = k_1 \cdot k_1 \cdot k_1$$

   To prove that k_1 is also a kernel, let

$$k_2 = f(x) = f(z) = 1$$
$$k_3 = (\frac{x}{\|x\|_2})^T(\frac{z}{\|z\|_2})$$

   Where k_2 is a kernel that always returns 1. If we assume k_3 is a kernel, then $k_2 + k_3$ satisfies rule ii and therefore k_1 is a kernel.

   k_3 is the dot product between x and z, satisfying rule iii. The terms x and z are linearly scaled by their norms, which is allowed by rule i. Since k_3 satisfies rules i and iii, it is also a kernel.

   Thus, all parts of the given expression are proved to adhere to the operation rules of a kernel, and

$$(1 + (\frac{x}{\|x\|_2})^T(\frac{z}{\|z\|_2}))^3$$

   is a kernel.

2 Kernelized SVM

1. Given $w^{(t)} = \sum_i^n a_i^{(t)} x_i$, and $K_j = \langle x_i, x_j \rangle$, we wish to show that

$$\langle w^{(t)},\, x_j \rangle = \langle x_i,\, x_j \rangle \cdot a^{(t)}$$

Expand inner products to summations

$$\sum_k^n w_k^{(t)} x_{jk} = \sum_k^n (x_{ik} \cdot x_{jk}) \cdot a^{(t)}$$

To make these expressions equal, we take:

$$w_k = x_{ik} \cdot a^{(t)}$$

For each component in $a^{(t)}$, notice that this is our definition of $w^{(t)}$:

$$w^{(t)} = \sum_i^n a_i^{(t)} x_i$$

Thus, it follows that

$$y_j \langle w^{(t)},\, x_j \rangle = y_j \cdot K_j \cdot a^{(t)}$$

2. Let $\alpha_{t+1}[i]$ act as a counter for the number of times example i in the training set has contributed to the loss. As in, it wasn't zero. Our update step, is to increment this value by 1 if:

$$y_i \frac{1}{\lambda t} \sum_j \alpha_t[j] y_i K(\boldsymbol{x}_i, \boldsymbol{x}_j) < 1$$

For a given sample i, we keep our regularization term $\frac{1}{\lambda t}$, and sum over all values j for i in our kernel matrix. If this is less than 1, then $\alpha_{t+1}[i]$ gets incremented by 1.

3. What about the case when the condition is ≥ 1? Then this step is similar to the non-kernalized version in which we do not update the weights – We do not update $\alpha_{t+1}[i_t]$. With this case, and the one in the previous question, we have handled both when there is, and is not, a margin violation. Thus, we can complete the kernalized Pegasos algorithm:

```
PROCEDURE Pegasos(S, λ, T, K) is

    SET α_1 = 0
    FOR t = 1 up to T do
        CHOOSE RANDOM i_t from {0, ..., length(S)}

        FOR ALL j ≠ i_t
            SET α_{t+1}[j] = α_t[j]

        IF y_{i_t} (1/(λt)) Σ_j α_t[j] y_{i_t} K(x_{i_t}, x_j) < 1 THEN

            SET α_{t+1}[i_t] = α_t[i_t] + 1
        ELSE
            SET α_{t+1}[i_t] = α_t[i_t]

RETURN α_{T+1}
```

3 Image Classification

1. Code for data loading and normalization

```python
import numpy as np
from sklearn.preprocessing import MinMaxScaler


def _load(filepath):
    x = []
    y = []
    with open(filepath, 'r') as fp:
        for line in fp:
            vec = [int(i) for i in line.split(',')]
            x.append(vec[1:])
            y.append(vec[0])

        return np.array(x), np.array(y)


def _scale(x):
    return MinMaxScaler(feature_range=(-1, 1)).fit_transform(x)


def load_train():
    x, y = _load("data/mnist_train.txt")
    return _scale(x), y


def load_test():
    x, y = _load("data/mnist_test.txt")
    return _scale(x), y
```

2. One vs All Class implementation

```python
import numpy as np
from sklearn.metrics import accuracy_score

from src.pegasos import Pegasos


class OneVAllClassifier:

    def __init__(self, lamb=2 ** -5):
        self.lamb = lamb

        # Dictionary of {target_class: [vector]}
        self.weight_vectors = {}

        # Key in self.weight_vectors
        self.best = None

        # TODO
        self.clf = Pegasos(lamb=lamb)

    @staticmethod
    def _relabel(y, target_value):
        return np.where(y == target_value, 1, -1)

    def _fit_one_append(self, X, y, target_class):
        yy = OneVAllClassifier._relabel(y, target_class)
        self.weight_vectors[target_class] = self.clf.fit(X, yy)

    def fit(self, X, y):
        # Fit a classifier and get the weights
        # for each class
        classes = np.unique(y)
        for c in classes:
            self._fit_one_append(X, y, c)

    def predict(self, X):
```

```
        # Array of (prediction_proba, class)
        y = []

        # Get prediction for each class
        for x in X:
            preds = []
            for c in self.weight_vectors.keys():
                w = self.weight_vectors[c]
                preds.append((np.dot(x, w), c))

            # Return the maximum
            best = max(preds)[1]
            y.append(best)

        print(y)

        return y

    def test(self, X, y):
        p = self.predict(X)
        return accuracy_score(y, p)
```

3. CV with 5 Folds implementation

```
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from src.one_v_all import OneVAllClassifier

def cv(X, y, lamb):
    print("Cross Validating Lambda:", lamb)
    n = 5
    acc = 0
    kf = KFold(n_splits=n)
    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        ova = OneVAllClassifier(lamb=lamb)
        ova.fit(X_train, y_train)

        # Add the accuracy score to the accumulator
        acc += ova.test(X_test, y_test)

    # Return average accuracy
    return acc / n

def plot_lambdas(cv_score, lambdas):
    plt.plot(lambdas, cv_score)
    plt.title("Cross Validated Avg. Accuracy vs. Lambda \nfor One Vs All Pegasos")
    plt.xlabel("Lambda for Pegasos (log2)")
    plt.ylabel("CV Average Accuracy (k = 5)")
    plt.show()

def find_best(X, y):
    lambas = [2 ** i for i in range(-5, 2)]
    cv_score = [cv(X, y, lamb=l) for l in lambas]

    # Plot the scores
    plot_lambdas(cv_score, lambas)
    return sorted(zip(cv_score, lambas), reverse=True)[0]
```

Plot of Lambda values. **Best Test Accuracy: 0.873 Best Lambda: 0.125**

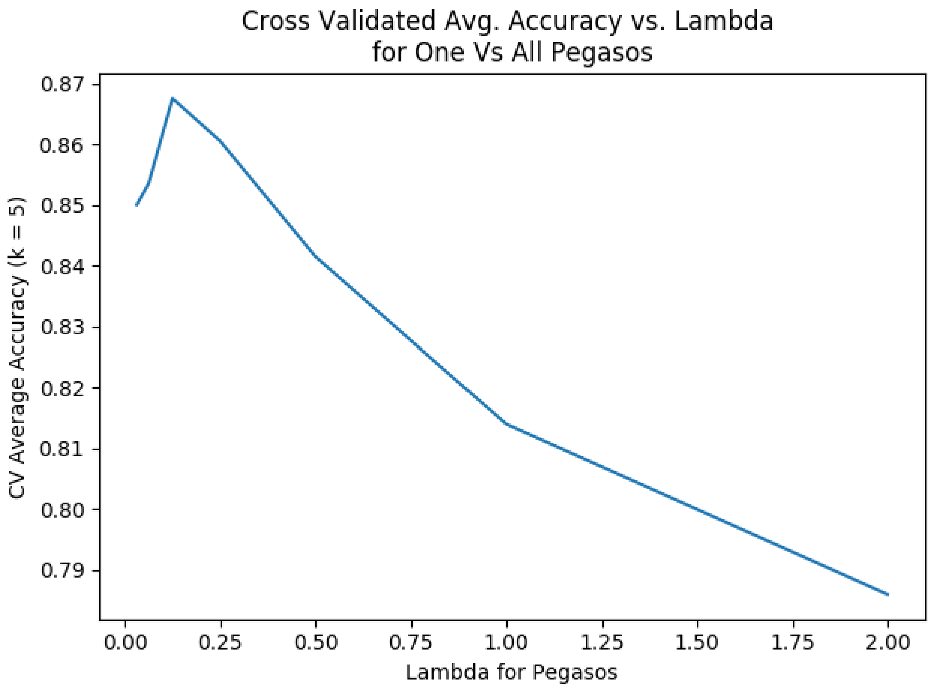

Figure 1: Log Lambda vs Average CV Accuracy of 5 Folds for Pegasos

4. Default SVC implementation. **Test Accuracy: 0.91**

```
def make_default_classifier(X_train, y_train, X_test, y_test):
    clf = SVC()
    ovr = OneVsRestClassifier(estimator=clf, n_jobs=-1)

    ovr.fit(X_train, y_train)

    # Evaluate
    acc = accuracy_score(y_test, ovr.predict(X_test))
    print(acc)

    return ovr
```

5. Default SVC 10-Fold CV implementation. **10-Fold CV Accuracy: 0.9055**

```
# Do vanilla CV on default values
n = 10
acc = 0
kf = KFold(n_splits=n)
for train_index, test_index in kf.split(X):
    X_tr, X_te = X_train[train_index], X_train[test_index]
    y_tr, y_te = y_train[train_index], y_train[test_index]

    ova = OneVsRestClassifier(estimator=SVC(), n_jobs=-1)
    ova.fit(X_tr, y_tr)

    # Add the accuracy score to the accumulator
    acc += accuracy_score(y_te, ova.predict(X_te))

# Return average accuracy
avg_cv_acc = acc / n

print("VANILLA CV SCORE:", avg_cv_acc)
```

6. Best SVC Grid Searcher with 10 Folds.
   Best CV Accuracy: 0.949 Best Test Accuracy: 0.949
   Parameters chosen: $\gamma = 0.007$, $C = 2.0$

```python
# Find BEST parameters using a grid search with 10 Folds
# By default  for this training set, SKLearn will use
# gamma = 1 / (n_features * X.var()) = 0.003353
params = {
    'C': [0.1, 0.5, 1.0, 1.5, 2.0],
    'gamma': ['auto', 'scale', 0.007, 0.01, 0.001]
}
grid = GridSearchCV(estimator=SVC(), param_grid=params, n_jobs=-1,
                    cv=n, verbose=2)
grid.fit(X_train, y_train)

print(grid.best_estimator_)
print(grid.best_params_)
print("OPTIMUM SCORE:", grid.best_score_)

return grid
```