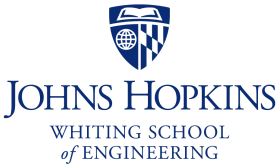
Supervised Learning



Background

Supervised machine learning is the problem of approximating a function $\mathbf{f}(x)$ that maps given inputs to given outputs. A function fit or a curve fit to the data corresponds to *learning* the data or *learning* the patterns that reside in the data. Different target functions and therefore different **hypothesis spaces** can be considered for a learning approach. The hypothesis space used by a machine learning system is the set of all hypotheses that might be returned by it. A learning problem is **realizable** if its hypothesis space contains the true function.

In general, supervised learning algorithms search the hypothesis space to find or pick the right hypothesis that will make good predictions for a given problem. It may be hard to find a hypothesis even if good hypotheses exist in the dataset for the given labels.

Hypotheses in Machine Learning

A hypothesis is a candidate explanation for a happening. A proper hypothesis is testable and can be either *true* or *false,* i.e. it is **falsifiable**. A proper hypothesis for learning fits the data and can be used to make predictions about new observations.

h is a single hypothesis, such as a particular model that maps inputs to outputs and can be evaluated to make predictions.
H is a hypothesis set, such as a searchable space of possible hypotheses for mapping inputs to outputs, and might be constrained by the learning problem, model, and configuration.

Supervised learning

Using pre-labeled datasets, algorithm learns how to predict the future test data, based on the following categories:

- Classification: Predicting whether a test sample belongs to one of the categorical classes. If there are only two classes, it's a binary classification problem. If an algorithm can only work with binary classes, then a set of N-choose-2 classifiers (i.e. $\binom{n}{2}$) are built - one for every class pair.
- Regression: Predicting a continuous numerical variable, such as a house price or a stock index.

In general, supervised learning minimizes an error (or maximizes a metric) where the error is the difference between the target output and the actual output (or the metric is the information gain of a variable, the reduction in entropy, etc.).

A **meta-classifier** is a classifier that makes a final prediction among all the predictions of other classifiers by using their predictions as its features. A meta-classifier is cascaded to classifier(s) that work on the dataset.

In supervised learning, the ground truth of the problem dataset has to be known so that data point labels can be tagged to be used by the ML algorithms. The following are basic supervised learning approaches.

Perceptron

The perceptron is a binary classifier, an M dimensional hyperplane w which separates the data points X into two categories $\{0, 1\}$, given by y.

$$\mathbf{f}(x) = \begin{cases} 1, & \text{if } w \cdot x + b > 0 \\ 0, & \text{otherwise} \end{cases}$$

In general, $x \in \mathbb{R}^M$, $X \in \mathbb{R}^{N \times M}$, $x_i = X_{i,*}$, $w \in \mathbb{R}^M$, $b \in \mathbb{R}$, $y \in \mathbb{R}^N$, where the dataset has N data points and M features (variables). The perceptron learning algorithm searches a w and b which minimizes the classification error E, i.e. the sum of differences between y_i and $\hat{y}_i = \mathbf{f}(x_i)$, for $i = 1, \ldots, N$.

The equation $w \cdot x + b$, or $w^\top x + b$ defines the hyperplane in M dimensions (i.e. the number of features, or columns of the X matrix).

If classes are not linearly separable, such as E never reaches to zero, then the algorithm will never stop, unless this situation is checked by the algorithm.

Note that the general perceptron algorithm is naturally a **greedy** type algorithm since there is no guarantee that the error or cost function $E = \sum_{i=1}^{N} |y - \hat{y}|$ is **convex** resulting in a global singular optimum. For greedy algorithms, if the optimization problem is non-convex, then a global minimum is not guaranteed to be found but instead a local optimum can be found.

A perceptron is the building block of a neural network.

Note that neural networks are built by stacking multiple perceptrons into layers and then cascading these layers into a network.

Naive Bayes

Naive Bayes classifier uses Bayes rule $P(c|x) = \dfrac{P(x|c)P(c)}{P(x)}$ or in other terms,

$$\text{Posterior prob} = \frac{(\text{Likelihood})(\text{Class prior prob})}{\text{Predictor prob}} \text{ ..... } (Eq.1)$$

Bayes rule assumes the features (predictors, columns of X) are **independent** of other variables (except for the dependent variable), and in general this is not true (thus the algorithm is called naive).

The algorithm computes posterior probabilities for each class c and picks the highest. It is called the Maximum A Posteriori (MAP) hypothesis, $\underset{c}{\texttt{argmax}} P(c|X)$.

Linear Regression

When numerical values are to be predicted for a dependent variable then the process is called regression. The counterpart is predicting class labels or categories for a dependent variable as in a supervised classification problem. The regression model output is continuous, non-nominal, real-valued. The process can be considered as a function approximation or a curve fit. Linear regression fits the data with the *best* line which goes through the input values.

The difference between the predicted point $\hat{y}$ (predicted `y_pred`) and the actual observation y (input `y`) is the **residue**. Finding the best line or hyperplane is an optimization problem, thus we need a cost function as in a perceptron. Such as,

$$\sum_i (\text{predicted}_i - \text{actual}_i)^2 = \sum_i (\text{residue}_i)^2$$

The learning algorithm minimizes the cost function to find the best line.

For the linear regression problem, where the line is given by $y = b_0 + b_1 x$, we are able to solve the optimization problem:

Minimize the cost C, where $C = \sum_i^N (\hat{y}_i - y_i)^2$ by taking the derivative, setting to zero and solving for b_0 and b_1. Then,

$$b_1 = \frac{N \sum xy - \sum x \sum y}{N \sum x^2 - (\sum x)^2} \text{ and } b_0 = \frac{\sum y - b_1 \sum x}{N}.$$

A perfect linear fit would have a total residue of 0. Generally, the input data would not result with a perfect line fit.

WATCH THE MODULE VIDEO for the Perceptron Visualization.

Example: Naive Bayes Classifier

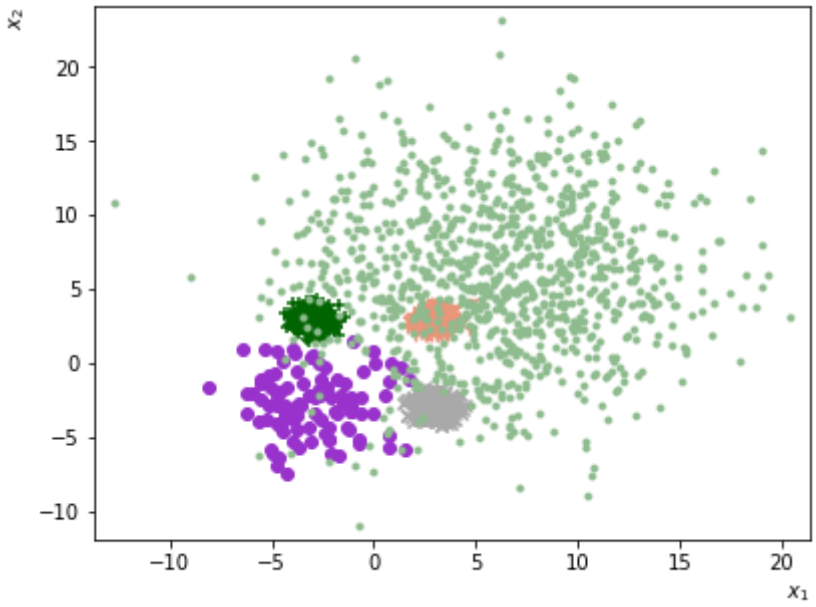
Let's code a Naive Bayes classifier from scratch.

Below implementation uses `scipy.stats.norm` Gaussian normal probability density function to compute the probabilities. Thus, numerical input features are required.

First, creation and visualization of the experimental dataset.

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams["figure.dpi"] = 72


# Following functions only work in 2-dimensions, i.e. X with 2 features
# And upto 5 clusters/classes

def get_minmax(_X, _m):  # _m = margin for visuals
    return _X[:,0].min()-_m, _X[:,0].max()+_m, _X[:,1].min()-_m, _X[:,1].max()+

def plotX(_X, _y, ax=None):  # Max 5 clusters/classes
    Colors = ['darkorchid', 'darkgreen', 'darkgrey', 'darksalmon', 'darkseagree
    Markers = ['o', '+', 'x', '1', '.']
    for c in range(max(_y)+1):
        plt.scatter(_X[_y==c,0], _X[_y==c,1], marker=Markers[c], color=Colors[c
    if ax is not None:
        ax.get_xaxis().set_ticks([])
        ax.get_yaxis().set_ticks([])

    x1_min, x1_max, x2_min, x2_max = get_minmax(_X, 1)
    plt.xlim(x1_min, x1_max)
    plt.ylim(x2_min, x2_max)
    plt.xlabel(r'$x_1$', horizontalalignment='right', x=1.0)
    plt.ylabel(r'$x_2$', horizontalalignment='right', y=1.0)
```

In [2]:
```
from sklearn.datasets import make_blobs

# 5 classes with 2 features
X1, y1 = make_blobs(n_samples=(100, 500, 500, 500, 1000), n_features=2, random_
                    cluster_std=[2, .5, .5, .5, 5],
                    centers=[(-3, -3), (-3, 3), (3, -3), (3, 3), (6, 6)])

plotX(X1, y1)
```



Following class has the following instance variables: `N` as the number of data points, `M` as the number of features, and `Cn` as the number of target classes. `Cn` is computed with `np.unique` on `y` where the class labels are expected to be integers between $[0, \mathrm{Cn})$.

Variables `N`, `M`, and `Cn` are set when a model is fit. A Predictor probability density function are computed for every feature m. A Prior and a Likelihood probability density function are computed for every class label c.

`np.bincount` counts each class label and returns the total count used for Priors.

After computing the Prior, Predictors, and the Likelihood, the resulting feature probabilities are multiplied to get $\mathrm{P} \longrightarrow$ the feature independence assumption of Naive Bayes algorithm. The class label which maximizes the probability P is picked as the predicted class label.

Note: Avoid *adding* (instead of multiplying) the feature probabilities, since the probabilistic model has M many independent variables.

In [3]:
```
# Build a Naive Bayes classifier from scratch for numerical features
#
from scipy.stats import norm
import numpy as np

class CustomNaiveBayes:  # Numerical features

    def __init__(self):
        self.N, self.M, self.Cn = None, None, None
        self.Prior, self.Predictors, self.Likelihood = None, None, None

    def fit(self, _Xtrain, _ytrain):
        self.N, self.M = _Xtrain.shape
        self.Cn = len(np.unique(_ytrain))  # Class numbers must be integer and
        assert min(_ytrain)==0 and max(_ytrain)==self.Cn-1  # Error check
        #
        # Prior probabilities - probability mass function
        self.Prior = np.bincount(_ytrain) / self.N
```

```
        #
        # Predictor probabilities
        # The predictors - list of (mu,std)
        self.Predictors = np.array([norm.fit(_Xtrain[:,_]) for _ in range(self.
        #
        # Likelihood probabilities
        # Dictionary of {list of (mu,std) per feature} per class
        self.Likelihood = np.empty((self.Cn,self.M,2))
        for c in range(self.Cn):
            self.Likelihood[c] = np.array([norm.fit(_Xtrain[np.where(_ytrain==c
        # Classifier model is composed of the data structures
        # Prior, Predictors, and Likelihood
        return self

    def predict(self, _Xtest):
        ypred = np.empty(len(_Xtest), dtype=np.int32)
        for i, x in enumerate(_Xtest):
            # Bayes rule based on training
            p_Likelihood = norm.pdf(x, self.Likelihood[:,:,0], self.Likelihood[
            p_Predictors = norm.pdf(x, self.Predictors[:,0], self.Predictors[:,
            # Take product of the feature probabilities - independence assumpti
            p = self.Prior*np.prod(p_Likelihood/p_Predictors, axis=1)  # Eq.1
            ypred[i] = np.argmax(p)

        return ypred
```

Sanity Check: Exercise a reclassification: Train with the entire dataset and test with the entire dataset on trained model. Expected performance should be high compared to a cross validation. Note that generally we do not report the **reclassification** performance.

In [4]:
```
%%time
from sklearn.metrics import accuracy_score

# Sanity check
clf2 = CustomNaiveBayes().fit(X1, y1)
y_pred = clf2.predict(X1)

print(f'Reclassification accuracy: {accuracy_score(y1, y_pred):.3f}')
```

```
Reclassification accuracy: 0.972
CPU times: total: 281 ms
Wall time: 295 ms
```

10-fold Cross Validation Performance: Compare the stratified cross validation performance with `sklearn.naive_bayes.GaussianNB` since the dataset classes are unbalanced. Accumulate statistics for `niter` times as each fold has a non-zero variance.

Compare the performance of our classifier `CustomNaiveBayes` to `sklearn.naive_bayes.GaussianNB`.

In [5]:
```
%%time
from sklearn.model_selection import StratifiedKFold
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

def eval_classifier(_clf, _X, _y, _niter, text=''):
    accs = []
```

```
    for i in range(_niter):
        kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=i)
        for train_index, test_index in kf.split(_X, _y):
            _clf.fit(_X[train_index], _y[train_index])
            ypred = _clf.predict(_X[test_index])
            accs += [accuracy_score(_y[test_index], ypred)]

    print(f'{text:<20s} Stratified 10-fold CV acc={np.mean(accs):.3f} with {_ni

eval_classifier(CustomNaiveBayes(), X1, y1, 10, 'CustomNaiveBayes')
eval_classifier(GaussianNB(), X1, y1, 10, 'sklearn GaussianNB')
eval_classifier(SVC(class_weight='balanced', kernel='linear', C=2), X1, y1, 10,
eval_classifier(SVC(class_weight='balanced', kernel='rbf', gamma=2, C=2), X1, y
```

```
CustomNaiveBayes     Stratified 10-fold CV acc=0.971 with 10 iterations
sklearn GaussianNB   Stratified 10-fold CV acc=0.971 with 10 iterations
SVM (linear)         Stratified 10-fold CV acc=0.862 with 10 iterations
SVM (rbf)            Stratified 10-fold CV acc=0.963 with 10 iterations
CPU times: total: 31.6 s
Wall time: 31.6 s
```

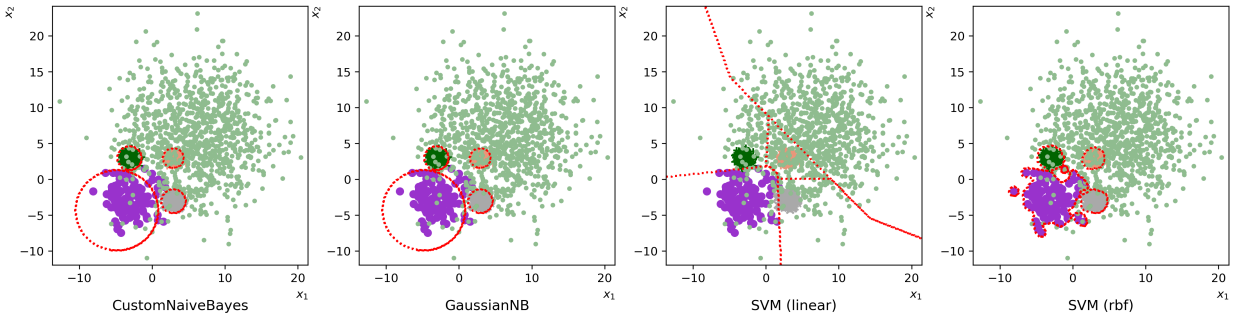
Decision Boundaries

Let's plot the model separating planes for each classifier. Notice that `CustomNaiveBayes` and the one from the `sklearn` library generate the same exact boundaries. Compare these to the linear SVM and RBF SVM classifiers and observe the SVM classifier boundaries are very different from the Naive Bayes.

In [6]:
```
def plot_decisionboundary(_X, _clf, _h, color_db='r'):  # _h = step size in the
    x1_min, x1_max, x2_min, x2_max = get_minmax(_X, 1)
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, _h), np.arange(x2_min, x2_
    Y = _clf.predict(np.c_[xx1.ravel(), xx2.ravel()]).reshape(xx1.shape)

    # debug
    # print(np.unique(Y,return_counts=True))

    plt.contour(xx1, xx2, Y, colors=color_db, linestyles='dotted')
```

In [7]:
```
%%time

h = 0.1  # mesh granularity of the plot

plt.figure(figsize=(18, 4), dpi=300)

plt.subplot(1, 4, 1)
plotX(X1, y1)
clf = CustomNaiveBayes().fit(X1, y1)
plot_decisionboundary(X1, clf, h)
plt.title('CustomNaiveBayes', y=-0.2)

plt.subplot(1, 4, 2)
plotX(X1, y1)
clf = GaussianNB().fit(X1, y1)
plot_decisionboundary(X1, clf, h)
plt.title('GaussianNB', y=-0.2)
```

```
plt.subplot(1, 4, 3)
plotX(X1, y1)
clf = SVC(class_weight='balanced', kernel='linear', C=2).fit(X1, y1)
plot_decisionboundary(X1, clf, h)
plt.title('SVM (linear)', y=-0.2)

plt.subplot(1, 4, 4)
plotX(X1, y1)
clf = SVC(class_weight='balanced', kernel='rbf', gamma=2, C=2).fit(X1, y1)
plot_decisionboundary(X1, clf, h)
plt.title('SVM (rbf)', y=-0.2)

plt.show()
```




```
CPU times: total: 25.9 s
Wall time: 25.9 s
```

Example: News Category Classification

Using `nltk` Tf-Idf features let's see how we would classify news categories in `nltk` Reuters corpus. Since Reuters have overlapping categories and in some cases multiple lists of categories, one possible approach can be repeating the data point for each category the news belongs to. Or as a second approach, we can fix the categories to a small number such as the 5 top news categories.

TF-IDF Metric as a Feature

The *weight, value, measure* of a term in a document is simply proportional to the term's frequency. However, this metric has to be *normalized* because the term frequency alone will tend to incorrectly emphasize documents which happen to use common and frequently used words (such as 'the', if stop-word filtered then 'news' in a news corpus), without giving enough weight to the more meaningful terms (such as 'alien' or 'inflation'). This normalization can be based on the number of the documents by counting how many documents have *that* term.

$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)$, where

$\mathrm{tf}(t, d) = \mathrm{frq}_{t,d}$, which is the raw count of term t in document d

$$\text{idf}(t, D) = \log \frac{N_d}{|\{d \in D : t \in d\}|}$$

$N_d = |D|$, total number of documents d in the corpus of D

Each unique word (i.e. one element of the vocabulary) in the documents will be represented in one column. Each row (data point) is generated from an input document. The data point can be a sentence, paragraph, section, and in this example it is an individual news text.

Classifier

In the following, we will use the pre-processed Tf-Idf as the input dataset and apply 10-fold cross validation to measure performance. The model will predict the news text category from a set of 6 categories.

In [8]:
```
from nltk.corpus import reuters
import pandas as pd

Documents = [reuters.raw(fid) for fid in reuters.fileids()]

# Categories are list of lists since each news may have more than 1 category
Categories = [reuters.categories(fid) for fid in reuters.fileids()]
CategoriesList = [_ for sublist in Categories for _ in sublist]
CategoriesSet = np.unique(CategoriesList)

print(f'N documents={len(Documents):d}, K unique categories={len(CategoriesSet)
```

```
N documents=10788, K unique categories=90
```

In [9]:
```
from collections import Counter

# Check the categories and their counts
counts = Counter(CategoriesList)
counts = sorted(counts.items(), key=lambda pair: pair[1], reverse=True)

print(counts[:10])
```

```
[('earn', 3964), ('acq', 2369), ('money-fx', 717), ('grain', 582), ('crude', 5
78), ('trade', 485), ('interest', 478), ('ship', 286), ('wheat', 283), ('cor
n', 237)]
```

Let's pick the top 5 categories and add the custom category named `other` assigned where if the news does not belong to any one of these 5 categories. Also, if a particular news text belongs to more than one category, then we will pick the highest occurring category for that news.

Note: As you may have noticed the problem at hand is a pretty unbalanced model development.

In [10]:
```
# Build the news category list
yCategories = [_[0] for _ in counts[:5]]
yCategories += ['other']

# Sanity check
print(f'K categories for classification= {len(yCategories):d} categories, {yCat
```

```
K categories for classification= 6 categories, ['earn', 'acq', 'money-fx', 'gr
ain', 'crude', 'other']
```

In [11]:
```
# Assign a category for each news text, including 'other'
yCat = []
for cat in Categories:
    bFound = False
    for _ in yCategories:
        if _ in cat:
            yCat += [_]
            bFound = True
            break  # So we add only one category for a news text
    if not bFound:
        yCat += ['other']

# Sanity check
print(f'N target categories={len(yCat):d}')
```

```
N target categories=10788
```

In [12]:
```
# Convert to numerical np.array which sklearn requires
ydocs = np.array([yCategories.index(_) for _ in yCat])
```

In [13]:
```
# StratifiedKFold will require indexable data structure
Docs = pd.Series(Documents)
Categories = pd.Series(yCat)

# Sanity check
print(Categories[0],'-->',Docs[0][:150],'\n',
      Categories[1],'-->',Docs[1][:150],'\n',
      Categories[2],'-->',Docs[2][:150])

# Size of the problem
print(f'N={len(Docs)} documents')
```

```
other --> ASIAN EXPORTERS FEAR DAMAGE FROM U.S.-JAPAN RIFT
  Mounting trade friction between the
  U.S. And Japan has raised fears among many of Asia's exportin
 grain --> CHINA DAILY SAYS VERMIN EAT 7-12 PCT GRAIN STOCKS
  A survey of 19 provinces and seven cities
  showed vermin consume between seven and 12 pct of Chin
 crude --> JAPAN TO REVISE LONG-TERM ENERGY DEMAND DOWNWARDS
  The Ministry of International Trade and
  Industry (MITI) will revise its long-term energy supply/
N=10788 documents
```

In [14]:
```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer

def kfold_eval_docs(_clf, _Xdocs, _ydocs):
    # Need indexable data structure
    accuracies = []
    kf = StratifiedKFold(n_splits=10, shuffle=False, random_state=None)
    for train_index, test_index in kf.split(_Xdocs, _ydocs):
        _clf.fit(_Xdocs[train_index], _ydocs[train_index])
        ypred = _clf.predict(_Xdocs[test_index])
        accuracies += [accuracy_score(_ydocs[test_index], ypred)]
```

```
    return np.array(accuracies)
```

In the following cells, let's see a variety of classifier models built by our dataset, mainly `Docs` and `Categories`. The Jupyter command `%%time` provides how long that cell runs, i.e. the classifier, in order to give a ballpark idea about their speed performances. A `Pipeline` is used to save memory where otherwise the computer will be overwhelmed by the size of the Tf-Idf matrix size. Note that `TfidfVectorizer` can set `max_features` by sorting the features and selecting the top frequencies. Slower algorithms like SVC can benefit lower number of features.

In [15]:
```
# Check the size of the dataset matrix X for this Tf-Idf feature extraction - i
from sklearn.feature_extraction.text import TfidfVectorizer

X_tfidf = TfidfVectorizer().fit_transform(Documents)
print(f'N data points= {X_tfidf.shape[0]}, M features= {X_tfidf.shape[1]}')
```

```
N data points= 10788, M features= 30916
```

In [16]:
```
# Use 1000 features instead of 30916

M_FEATURES= 1000
```

In [17]:
```
%%time
from sklearn.svm import SVC

svm_rbf = Pipeline([('vect', CountVectorizer(max_features=M_FEATURES)),
                    ('tfidf', TfidfTransformer()),
                    ('clf', SVC(kernel='rbf', gamma='scale', class_weight='bala
                   ])
acc = kfold_eval_docs(svm_rbf, Docs, Categories)
print(f'Support Vector Machine (RBF) CV accuracy={np.mean(acc):.3f} {np.std(acc
```

```
Support Vector Machine (RBF) CV accuracy=0.936 0.008
CPU times: total: 2min 25s
Wall time: 2min 25s
```

In [18]:
```
%%time
from sklearn.svm import LinearSVC

svm_lin = Pipeline([('vect', CountVectorizer(max_features=M_FEATURES)),
                    ('tfidf', TfidfTransformer()),
                    ('clf', LinearSVC(class_weight='balanced'))
                   ])
acc = kfold_eval_docs(svm_lin, Docs, Categories)
print(f'Support Vector Machine (linear SVC) CV accuracy={np.mean(acc):.3f} {np.
```

```
Support Vector Machine (linear SVC) CV accuracy=0.935 0.007
CPU times: total: 7.58 s
Wall time: 7.59 s
```

Question: Compare the model performance and model building speed of RBF SVM (`SVC` module), and the `LinearSVC` module. Why do you think `LinearSVC` works faster with higher performance (which also scales nicely with the dataset size)?

In [19]:
```
%%time
from sklearn.naive_bayes import MultinomialNB

nb = Pipeline([('vect', CountVectorizer(max_features=M_FEATURES)),
               ('tfidf', TfidfTransformer()),
               ('clf', MultinomialNB())
              ])
acc = kfold_eval_docs(nb, Docs, Categories)
print(f'Naive Bayes CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
```

```
Naive Bayes CV accuracy=0.879 0.022
CPU times: total: 6.39 s
Wall time: 6.4 s
```

In [20]:
```
%%time
from sklearn.ensemble import RandomForestClassifier

n_cores = 8
rf = Pipeline([('vect', CountVectorizer(max_features=M_FEATURES)),
               ('tfidf', TfidfTransformer()),
               ('clf', RandomForestClassifier(n_jobs=n_cores, n_estimators=300,
                                              max_depth=10, random_state=0, cla
              ])
acc = kfold_eval_docs(rf, Docs, Categories)
print(f'Random Forest CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
```

```
Random Forest CV accuracy=0.840 0.022
CPU times: total: 1min 9s
Wall time: 15.8 s
```

In [21]:
```
%%time
import warnings
from sklearn.exceptions import ConvergenceWarning
from sklearn.linear_model import LogisticRegression

# To avoid non-convergence one has to increase 'max_iter' parameter
lr = Pipeline([('vect', CountVectorizer(max_features=M_FEATURES)),
               ('tfidf', TfidfTransformer()),
               ('clf', LogisticRegression(solver='lbfgs', multi_class='auto', m
              ])
with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=ConvergenceWarning)
    acc = kfold_eval_docs(lr, Docs, Categories)

print(f'Logistic Regression CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
```

```
Logistic Regression CV accuracy=0.925 0.009
CPU times: total: 24.5 s
Wall time: 11.3 s
```

Example: Programming Language Detection

Once more, using Tf-Idf features let's see how we would classify the programming language given the text.

Similar to previous approach, we will use the pre-processed Tf-Idf as the input dataset, but this time we can apply 70-30 split validation to measure performance since the dataset is pretty big.

The dataset is composed of Stackoverflow website (https://stackoverflow.com/) posts about 20 different programming languages.

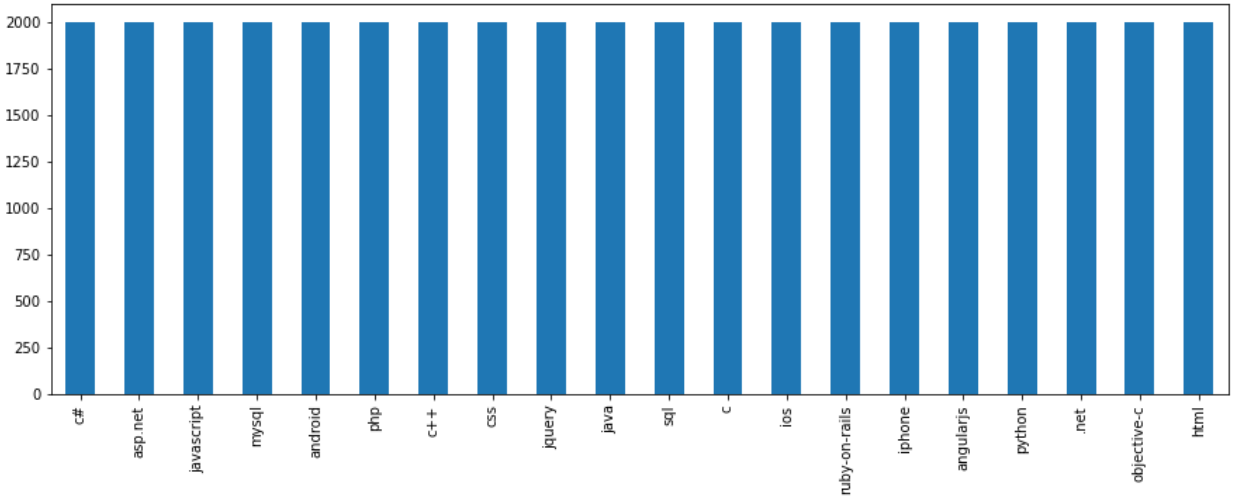
Note that Stackoverflow website has forums where users post programming language questions. Generally a question is related to one programming language. Problem, given the user question text can we classify the question to text one of 20 different programming languages - to help our website.

In [22]:
```
import pandas as pd

# Load the data
df = pd.read_csv('../../../EP_datasets/stack_overflow_data.csv')
df = df[pd.notnull(df['tags'])]  # filter out not known labels

# Sanity check
print(df.head(20))
print(f"Number of words= {df['post'].apply(lambda x: len(x.split(' '))).sum()}'
```

```
                                                 post           tags
0   what is causing this behavior  in our c# datet...             c#
1   have dynamic html load as if it was in an ifra...        asp.net
2   how to convert a float value in to min:sec  i ...    objective-c
3   .net framework 4 redistributable  just wonderi...           .net
4   trying to calculate and print the mean and its...         python
5   how to give alias name for my website  i have ...        asp.net
6   window.open() returns null in angularjs  it wo...      angularjs
7   identifying server timeout quickly in iphone  ...         iphone
8   unknown method key  error in rails 2.3.8 unit ...  ruby-on-rails
9   from the include  how to show and hide the con...      angularjs
10  when we need interface c# <blockquote>    <str...             c#
11  how to install .ipa on jailbroken iphone over ...            ios
12  dynamic textbox text - asp.net  i m trying to ...        asp.net
13  rather than bubblesorting these names...the pr...              c
14  site deployed in d: drive and uploaded files a...        asp.net
15  connection in .net  i got     <blockquote>    ...           .net
16  how to subtract 1 from an int  how do i subtra...    objective-c
17  ror console show syntax error  i want to add d...  ruby-on-rails
18  distance between 2 or more drop pins  i was do...         iphone
19  sql query - how to exclude a record from anoth...            sql
Number of words= 10286120
```

In [23]:
```
# Check the number of data points in each category - balanced or not
plCategories = np.unique(df['tags'])
print(f'K categories={len(plCategories):d} {plCategories}')

plt.figure(figsize=(15,5), dpi=72)
df.tags.value_counts().plot(kind='bar');
```

```
K categories=20 ['.net' 'android' 'angularjs' 'asp.net' 'c' 'c#' 'c++' 'css'
'html' 'ios'
 'iphone' 'java' 'javascript' 'jquery' 'mysql' 'objective-c' 'php'
 'python' 'ruby-on-rails' 'sql']
```



In [24]:
```
plCategories_mapping = {k:i for i, k in enumerate(plCategories)}

# Convert to numerical np.array which sklearn requires
yposts = np.array([plCategories_mapping[_] for _ in df['tags']])
```

Let's see how many features we have in our Tf-Idf matrix.

In [25]:
```
# Check counts, size of the X dataset - raw features
X_tfidf = TfidfVectorizer().fit_transform(df.post)
print(f'N data points= {X_tfidf.shape[0]}, M features= {X_tfidf.shape[1]}')
```

```
N data points= 40000, M features= 181015
```

In [26]:
```
M_FEATURES= 3000

def make_pipeline(_clf, _mx_feats):
    return Pipeline([('vect', CountVectorizer(max_features=_mx_feats)),
                     ('tfidf', TfidfTransformer()), ('clf', _clf)])

svm_rbf = make_pipeline(SVC(kernel='rbf', gamma='scale', class_weight='balanced
svm_lin = make_pipeline(LinearSVC(class_weight='balanced'), M_FEATURES)
nb = make_pipeline(MultinomialNB(), M_FEATURES)
rf = make_pipeline(RandomForestClassifier(n_jobs=n_cores, n_estimators=300,
    max_depth=10, random_state=0, class_weight='balanced'), M_FEATURES)
lr = make_pipeline(LogisticRegression(solver='lbfgs', multi_class='auto', max_i
    class_weight='balanced'), M_FEATURES)
```

In [27]:
```
%%time

acc = kfold_eval_docs(nb, df.post, df.tags)
print(f'Naive Bayes CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
```

```
Naive Bayes CV accuracy=0.731 0.005
CPU times: total: 29.5 s
Wall time: 29.5 s
```

In [28]:
```
%%time

acc = kfold_eval_docs(rf, df.post, df.tags)
print(f'Random Forest CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
```

```
Random Forest CV accuracy=0.768 0.007
CPU times: total: 3min 51s
Wall time: 56.3 s
```

In [29]:
```
%%time

acc = kfold_eval_docs(svm_lin, df.post, df.tags)
print(f'Support Vector Machine CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}
```

```
Support Vector Machine CV accuracy=0.795 0.006
CPU times: total: 46.1 s
Wall time: 46.1 s
```

In [30]:
```
%%time

acc = kfold_eval_docs(lr, df.post, df.tags)
print(f'Logistic Regression CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
```

```
Logistic Regression CV accuracy=0.802 0.006
CPU times: total: 13min 15s
Wall time: 2min 4s
```

Note: SVM with an RBF kernel cross validation would even take a longer time to finish on my machine. Now for demonstration, let's see a 70-30 split-evaluation to examine performances or 20-class problem. Recall that CV runs the training-testing iterations for 10 times but split-eval will run only once.

In [31]:
```
def split_eval_docs(_clf, _Xdocs, _ydocs):
    X_train, X_test, y_train, y_test = train_test_split(_Xdocs, _ydocs, test_si
    _clf.fit(X_train, y_train)
    ypred = _clf.predict(X_test)
    return y_test, ypred
```

In [32]:
```
%%time

y_test, y_pred = split_eval_docs(nb, df.post, df.tags)
print('Naive Bayes\n' + classification_report(y_test, y_pred, target_names=plCa
```

```
Naive Bayes
                precision    recall  f1-score   support

          .net       0.52      0.67      0.59       589
       android       0.96      0.79      0.87       661
     angularjs       0.94      0.90      0.92       606
       asp.net       0.74      0.69      0.71       613
             c       0.74      0.86      0.80       601
            c#       0.64      0.58      0.60       585
           c++       0.84      0.71      0.77       621
           css       0.70      0.83      0.76       587
          html       0.57      0.63      0.60       560
           ios       0.63      0.60      0.62       611
        iphone       0.59      0.61      0.60       593
          java       0.81      0.76      0.78       581
    javascript       0.77      0.62      0.68       608
        jquery       0.71      0.76      0.73       593
         mysql       0.66      0.74      0.70       592
   objective-c       0.67      0.66      0.67       597
           php       0.80      0.71      0.75       604
        python       0.82      0.88      0.85       610
ruby-on-rails       0.93      0.87      0.90       595
           sql       0.69      0.75      0.72       593

      accuracy                           0.73     12000
     macro avg       0.74      0.73      0.73     12000
  weighted avg       0.74      0.73      0.73     12000

CPU times: total: 3.12 s
Wall time: 3.1 s
```

In [33]:
```
%%time

y_test, y_pred = split_eval_docs(rf, df.post, df.tags)
print('Random Forest\n' + classification_report(y_test, y_pred, target_names=pl
```

```
Random Forest
                precision    recall  f1-score   support

          .net       0.42      0.70      0.53       589
       android       0.95      0.86      0.91       661
     angularjs       0.97      0.97      0.97       606
       asp.net       0.83      0.72      0.77       613
             c       0.78      0.82      0.80       601
            c#       0.48      0.55      0.51       585
           c++       0.96      0.59      0.73       621
           css       0.73      0.89      0.80       587
          html       0.77      0.49      0.60       560
           ios       0.68      0.67      0.67       611
        iphone       0.72      0.58      0.64       593
          java       0.84      0.81      0.82       581
    javascript       0.73      0.80      0.76       608
        jquery       0.84      0.84      0.84       593
         mysql       0.77      0.89      0.83       592
   objective-c       0.72      0.68      0.70       597
           php       0.86      0.84      0.85       604
        python       0.92      0.91      0.92       610
ruby-on-rails       0.95      0.94      0.94       595
           sql       0.76      0.81      0.78       593

      accuracy                           0.77     12000
     macro avg       0.78      0.77      0.77     12000
  weighted avg       0.79      0.77      0.77     12000

CPU times: total: 19.7 s
Wall time: 5.33 s
```

In [34]:
```
%%time

y_test, y_pred = split_eval_docs(svm_lin, df.post, df.tags)
print('SupportVector Machine\n' + classification_report(y_test, y_pred, target_
```

```
SupportVector Machine
                precision    recall  f1-score   support

          .net       0.70      0.66      0.68       589
       android       0.91      0.89      0.90       661
     angularjs       0.98      0.97      0.98       606
       asp.net       0.80      0.77      0.78       613
             c       0.80      0.85      0.83       601
            c#       0.59      0.61      0.60       585
           c++       0.80      0.73      0.77       621
           css       0.79      0.85      0.82       587
          html       0.66      0.68      0.67       560
           ios       0.67      0.65      0.66       611
        iphone       0.67      0.67      0.67       593
          java       0.83      0.82      0.83       581
    javascript       0.79      0.78      0.78       608
        jquery       0.85      0.86      0.85       593
         mysql       0.83      0.80      0.82       592
   objective-c       0.68      0.65      0.67       597
           php       0.84      0.84      0.84       604
        python       0.91      0.93      0.92       610
ruby-on-rails       0.94      0.94      0.94       595
           sql       0.77      0.86      0.81       593

      accuracy                           0.79     12000
     macro avg       0.79      0.79      0.79     12000
  weighted avg       0.79      0.79      0.79     12000

CPU times: total: 4.27 s
Wall time: 4.26 s
```

In [35]:
```
%%time

y_test, y_pred = split_eval_docs(lr, df.post, df.tags)
print('Logistic Regression\n' + classification_report(y_test, y_pred, target_na
```

```
Logistic Regression
               precision    recall  f1-score   support

         .net       0.70      0.70      0.70       589
      android       0.95      0.86      0.90       661
    angularjs       0.99      0.93      0.96       606
      asp.net       0.80      0.74      0.77       613
            c       0.79      0.85      0.82       601
           c#       0.60      0.66      0.63       585
          c++       0.84      0.73      0.78       621
          css       0.80      0.86      0.83       587
         html       0.67      0.74      0.70       560
          ios       0.70      0.69      0.69       611
       iphone       0.67      0.71      0.69       593
         java       0.85      0.82      0.84       581
   javascript       0.81      0.78      0.79       608
       jquery       0.85      0.84      0.84       593
        mysql       0.83      0.81      0.82       592
  objective-c       0.72      0.69      0.70       597
          php       0.87      0.84      0.86       604
       python       0.90      0.94      0.92       610
ruby-on-rails       0.96      0.92      0.94       595
          sql       0.77      0.88      0.82       593

     accuracy                           0.80     12000
    macro avg       0.80      0.80      0.80     12000
 weighted avg       0.80      0.80      0.80     12000

CPU times: total: 59.8 s
Wall time: 10.1 s
```

Important

Generally Tf-Idf vectorizer generates a sparse matrix (which can be obtained by `todense()` method), hindering some general usages. And note that since the number of features M is around 180k, the dense X matrix may be too big for common computer memory, e.g. 16 GB. We have to process more to reduce the number of words (or columns), using techniques such as Natural Language Processing. Note that we have been already using `Pipeline` to save memory and speed up computations.

Final Words

As can be seen from the model performances and run-time durations, `LinearSVC` is the best classifier for such text classification problems. One of the reasons is that Tf-Idf feature matrix is very suitable for LinearSVC which places a hyperplane (surface) between the classes. Because the number of dimensions/features M is very high (in fact higher than N), linear SVM works very good. Recall the raw Tf-Idf matrix had 181015 features. Since $N << M$ linear SVM works very well.

References

1. Raschka, Sebastian. Python Machine Learning Ed. 3. Packt Publishing, 2019.
2. Platt, John. "Sequential minimal optimization: A fast algorithm for training support vector machines." (1998).

Exercises

Exercise 1. In the Decision Boundaries cell `7`, change the SVM RBF gamma parameter to 0.5. Note your observations. Change the gamma to 10.0 and note your observations. What happened to the decision boundary and why?

Exercise 2. In the News Category Classification section, increase the number of categories from top-5 to top-10. Use a Random Forest classifier and report 10-fold CV accuracy. Which class/topic is the most accurately predicted? (Hint: Use a confusion matrix)

Exercise 3. Remove the `max_features` to see the effect of full set of features. Why do you think the performance increases but very little? Now set `max_features` to a low number, do you think there is a chance that some documents will have a zero feature vector, i.e. none of the words from the feature vocabulary exists?
